MybatisPlus快速入门

简介

MyBatis-Plus (opens new window)（简称 MP）是一个 MyBatis (opens new window)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

- **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 CURD，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

快速入门

地址：https://baomidou.com/guide/quick-start.html

使用第三方组件

1. 导入对应的依赖
2. 研究依赖如何配置
3. 代码如何编写
4. 提高扩展技术能力

这是3.4.1版本的与低版本有所区别

> 步骤

1. 创建数据库 `mybatis_plus`
2. 创建 `user` 表

```
DROP TABLE IF EXISTS user;

CREATE TABLE user
(
    id BIGINT(20) NOT NULL COMMENT '主键ID',
    name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',
    age INT(11) NULL DEFAULT NULL COMMENT '年龄',
    email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',
    PRIMARY KEY (id)
);
#真实开发中，version(乐观锁)、deleted(逻辑删除)、gmt_create/gmt_modified
```

3. 初始化工程 `springboot`

   添加依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>com.baomidou</groupId>
```

```
        <artifactId>mybatis-plus-boot-
starter</artifactId>
        <version>3.4.1</version>
    </dependency>
</dependencies>
```

4. 配置

   在 `application.yml` 配置文件中添加 mysql的相关配置：

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    #mysql8 要写时区
    url: jdbc:mysql://localhost:3306/springboot?
serverTimezone=Asia/Shanghai&useUnicode=true&charact
erEncoding=utf-8
    username: root
    passwor: root
```

5. 在 Spring Boot 启动类中添加 `@MapperScan` 注解，扫描 Mapper 文件夹：

```
@SpringBootApplication
//注意只写到mapper包一级就可以了，不要写到子包
@MapperScan("com.mycode.mapper")
public class MybatisPlusApplication {

    public static void main(String[] args) {

 SpringApplication.run(MybatisPlusApplication.class,
args);
    }

}
```

6. 编写pojo类

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    //private Integer id;
    private Long id;//采坑：数据库用的是BigInt要使用
Long才行！！！
    private String name;
    private Integer age;
    private String email;

}
```

7. 编写mapper类

```
@Repository
public interface UserMapper extends BaseMapper<User>
{
}
```

8. 编写测试类

```
@SpringBootTest
class MybatisPlusApplicationTests {

    @Autowired
    private UserMapper userMapper;

    @Test
    void contextLoads() {
        List<User> users =
userMapper.selectList(null);
        users.forEach(System.out::println);
    }

}
```

9. 结果

```
                        3.4.1
2020-12-16 23:44:21.392  INFO 21176 --- [           main] com.mycode.MybatisPlusApplicationTests
2020-12-16 23:44:21.648  INFO 21176 --- [           main] com.zaxxer.hikari.HikariDataSource
2020-12-16 23:44:21.828  INFO 21176 --- [           main] com.zaxxer.hikari.HikariDataSource
User(id=1, name=Jone, age=18, email=test1@baomidou.com)
User(id=2, name=Jack, age=20, email=test2@baomidou.com)
User(id=3, name=Tom, age=28, email=test3@baomidou.com)
User(id=4, name=Sandy, age=21, email=test4@baomidou.com)
User(id=5, name=Billie, age=24, email=test5@baomidou.com)
2020-12-16 23:44:21.879  INFO 21176 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource
2020-12-16 23:44:21.884  INFO 21176 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource
2020-12-16 23:44:21.885  INFO 21176 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor
```

配置日志

在application.yml中添加一条配置

```yaml
mybatis-plus:
  configuration:
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
```

结果

CURD

insert

- 主键生成策略

@IdType(opens new window)

值	描述
`AUTO`	数据库ID自增 数据库表必须设置自增长
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`)

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    private String name;
    private Integer age;
    private String email;

}
```

```java
@Test
void contextLoads() {
    User user = new User();
    user.setName("张三");
    user.setAge(20);
    user.setEmail("zhangsan@gmail.com");
    int insert = userMapper.insert(user);
    System.out.println(insert);
}
```

update

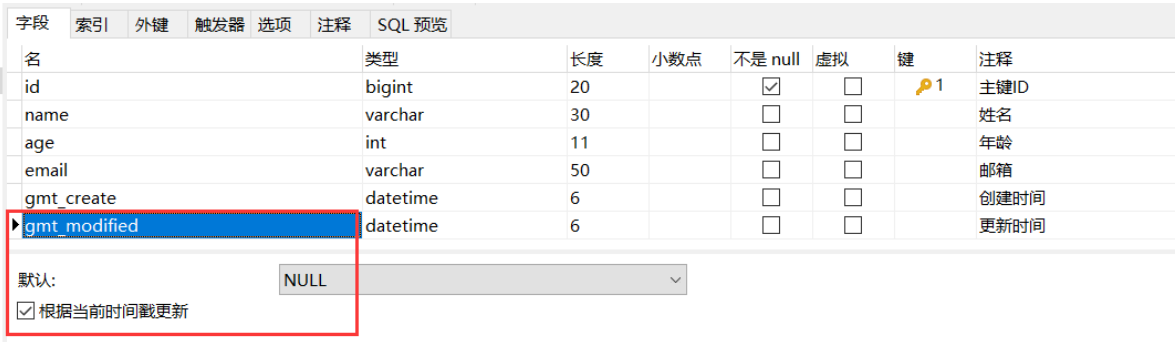
```java
@Test
public void updateTest(){
    User user = new User();
    user.setId(5L);
    user.setName("张三");
    user.setAge(1);
    user.setEmail("zhangsan@gmail.com");
    userMapper.updateById(user);
}
```

自动填充

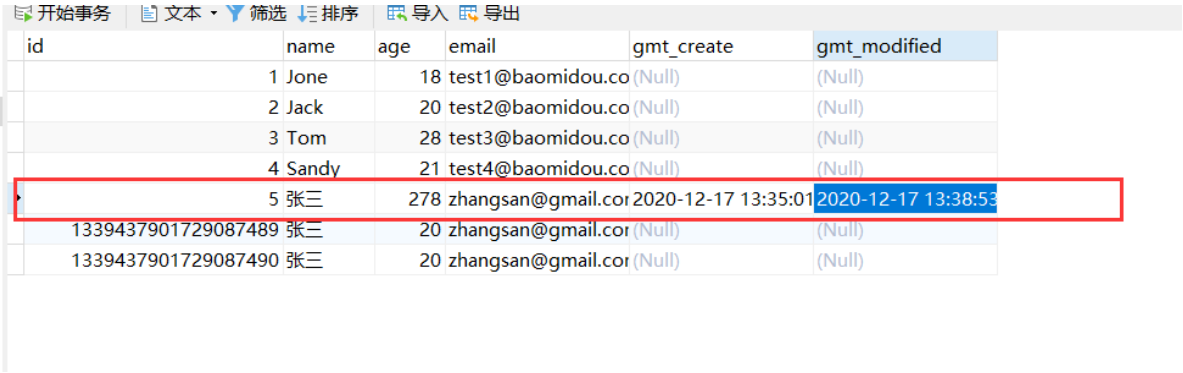
创建时间、修改时间！自动完成、不要手动更新。 `gmt_create`、`gmt_modified`几乎所有的表都要配置上

> 方式一：数据库级别

新增两个字段`gmt_create`、 `gmt_modified`




```
@Test
public void updateTest(){
    User user = new User();
    user.setId(5L);
    user.setName("张三");
    user.setAge(278);
    user.setEmail("zhangsan@gmail.com");
    userMapper.updateById(user);
}
```



坑：多次执行该代码不会执行时间戳更新 具体原因不清楚

> 方式二：代码级别

方法一不推荐使用

使用注解

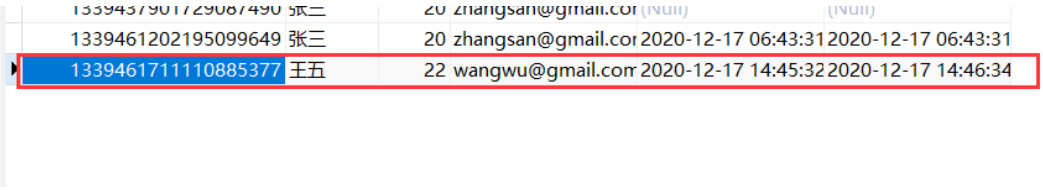
```
//字段添加填充内容
@TableField(fill = FieldFill.INSERT)//在插入的时候填充
private Date gmtCreate;
@TableField(fill = FieldFill.INSERT_UPDATE)//在插入和更新的的时候填充
private Date gmtModified;
```

```
@Slf4j
@Component //一定要把处理器放到IOC容器中
public class MeMetaObjectHandler implements MetaObjectHandler {

    //插入时的填充策略
    @Override
    public void insertFill(MetaObject metaObject) {
      log.info("start insert fill");
      this.setFieldValByName("gmtCreate", new Date(), metaObject);
      this.setFieldValByName("gmtModified", new Date(), metaObject);
    }

    //更新时的填充数据
    @Override
    public void updateFill(MetaObject metaObject) {
        this.setFieldValByName("gmtModified", new Date(), metaObject);
    }
}
```

执行插入和更新操作之后:



1339461202195099649	张三	20	zhangsan@gmail.cor	2020-12-17 06:43:31	2020-12-17 06:43:31
1339461711110885377	王五	22	wangwu@gmail.com	2020-12-17 14:45:32	2020-12-17 14:46:34

乐观锁

> 乐观锁：开放，不会出现问题，出现问题再次更新值
>
> 悲观锁：悲观，都要上锁，再去操作

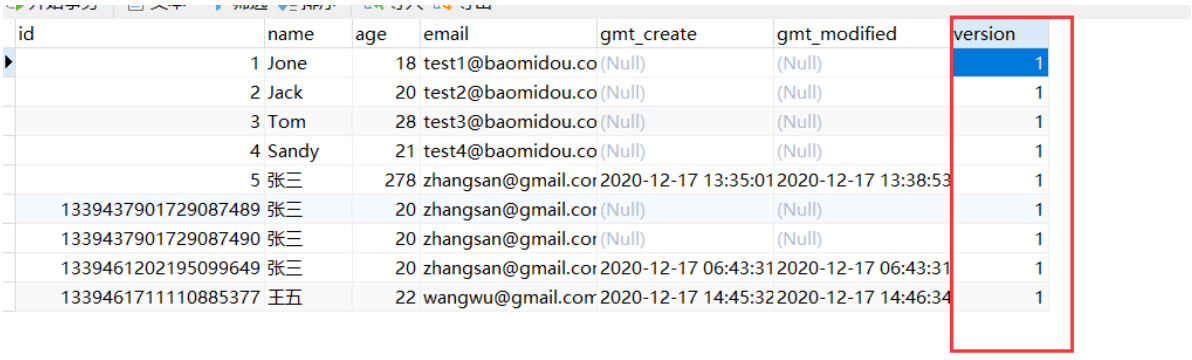
乐观锁

- 取出记录时，获取当前version
- 更新时，带上version
- 执行更新时，set version = newVersion where version = version + 1
- 如果version不对，就更新失败

```
-- 初始version为1
update user set name='zhangsan',version = version + 1
where id = 1 and version = 1;
```

> 实现乐观锁

1. 修改表及其实体类 添加version字段



id	name	age	email	gmt_create	gmt_modified	version
1	Jone	18	test1@baomidou.co	(Null)	(Null)	1
2	Jack	20	test2@baomidou.co	(Null)	(Null)	1
3	Tom	28	test3@baomidou.co	(Null)	(Null)	1
4	Sandy	21	test4@baomidou.co	(Null)	(Null)	1
5	张三	278	zhangsan@gmail.cor	2020-12-17 13:35:01	2020-12-17 13:38:53	1
1339437901729087489	张三	20	zhangsan@gmail.cor	(Null)	(Null)	1
1339437901729087490	张三	20	zhangsan@gmail.cor	(Null)	(Null)	1
1339461202195099649	张三	20	zhangsan@gmail.cor	2020-12-17 06:43:31	2020-12-17 06:43:31	1
1339461711110885377	王五	22	wangwu@gmail.com	2020-12-17 14:45:32	2020-12-17 14:46:34	1

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
```

```
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    @TableField(fill = FieldFill.INSERT)
    private Date gmtCreate;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date gmtModified;


    @Version
    private Integer version;//添加的version字段
}
```

2. 配置乐观锁类

```
//更新新版应该这样使用
@Configuration
@MapperScan("com.mycode.mapper")
@EnableTransactionManagement
public class MyBatisPlusConfig {

    /**
     * 乐观锁
     * 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)
     */
    @Bean
    public MybatisPlusInterceptor
mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        //乐观锁插件配置
```

```
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        return interceptor;
    }

    @Bean
    public ConfigurationCustomizer
configurationCustomizer() {
        return configuration ->
configuration.setUseDeprecatedExecutor(false);
    }

}

/*@Configuration
@MapperScan("com.mycode.mapper")
@EnableTransactionManagement
public class MyBatisPlusConfig {
    //OptimisticLockerInterceptor 注意这是一个过时类但是用
新的会报错，暂时先用旧的
    @Bean
    public OptimisticLockerInterceptor
optimisticLockerInterceptor(){
        return new OptimisticLockerInterceptor();
    }
}*/

/*
这种用法是错误的
@Configuration
@MapperScan("com.mycode.mapper")
@EnableTransactionManagement
public class MyBatisPlusConfig {

    @Bean
    public OptimisticLockerInnerInterceptor
optimisticLockerInterceptor(){
```

```
        return new OptimisticLockerInnerInterceptor();
    }
}*/
```

执行测试：

```
@Test
public void OptimisticLockerTest(){
    User user =
userMapper.selectById(1339461711110885377L);
    System.out.println(user);
    user.setName("王麻子");
    user.setAge(53);
    userMapper.updateById(user);
}
```

注意：必须先查询再更新才能使用乐观锁，直接更新是不会使用乐观锁

多线程测试：

```
@Test
public void OptimisticLockerTest2(){
    //模拟多线程
    //第一个线程先查询
    User user =
userMapper.selectById(1339461711110885377L);
    System.out.println(user);
    user.setName("王麻子");
    user.setAge(53);
    //第二个多线程再执行查询并先更新
```

```
    User user2 =
userMapper.selectById(1339461711110885377L);
    System.out.println(user2);
    user2.setName("王麻子大爷");
    user2.setAge(99);
    userMapper.updateById(user2);
    //最后更新第一个线程会失败
    //可以使用自旋锁
    userMapper.updateById(user);
}
```

select

```
//批量查询
@Test
public void selectTest(){
    List<User> users =
userMapper.selectBatchIds(Arrays.asList(1, 2, 3, 4));
    users.forEach(System.out::println);
}
```

```
//条件查询
@Test
public void selectByMap(){
    Map<String, Object> map = new HashMap<>();
    map.put("name", "jack");
    List<User> users = userMapper.selectByMap(map);
    System.out.println(users);
}
```

分页查询

内置分页插件

1. 配置

```
@Bean
    public MybatisPlusInterceptor
mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        //乐观锁
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        //分页插件
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor());
        //......直接在这里面new对应的插件

        return interceptor;
    }
```

插件

- 自动分页: PaginationInnerInterceptor
- 多租户: TenantLineInnerInterceptor
- 动态表名: DynamicTableNameInnerInterceptor
- 乐观锁: OptimisticLockerInnerInterceptor
- sql性能规范: IllegalSQLInnerInterceptor
- 防止全表更新与删除: BlockAttackInnerInterceptor

2. 测试

```java
//分页查询
@Test
public void pageTest(){
    Page<User> page = new Page<>(2, 5);//参数一 当前页 参数二 页面大小
    userMapper.selectPage(page, null);
    page.getRecords().forEach(System.out::println);
}
```

```
JDBC Connection [HikariProxyConnection@222882200 wrapping com.mysql.cj.jdbc.ConnectionImpl@7cfac39f] will not be managed by Spring
==>  Preparing: SELECT COUNT(*) FROM user
==> Parameters:
<==    Columns: COUNT(*)
<==        Row: 9
<==      Total: 1
==>  Preparing: SELECT id,name,age,email,gmt_create,gmt_modified,version FROM user LIMIT ?,?
==> Parameters: 5(Long), 5(Long)
<==    Columns: id, name, age, email, gmt_create, gmt_modified, version
<==        Row: 1339437901729087489, 张三, 20, zhangsan@gmail.com, null, null, 1
<==        Row: 1339437901729087490, 张三, 20, zhangsan@gmail.com, null, null, 1
<==        Row: 1339461202195099649, 张三, 20, zhangsan@gmail.com, 2020-12-17 06:43:31.247000, 2020-12-17 06:43:31.247000, 1
<==        Row: 1339461711110885377, 王麻子大爷, 99, tianliu@gmail.com, 2020-12-17 14:45:32.583000, 2020-12-17 16:24:18.485000, 4
<==      Total: 4
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6af91cc8]
```

delete

.....

逻辑删除

1. 修改表和实体类

id	name	age	email	gmt_create	gmt_modified	version	deleted
1	Jone	18	test1@baomidou.co	(Null)	(Null)	1	0
2	Jack	20	test2@baomidou.co	(Null)	(Null)	1	0
3	Tom	28	test3@baomidou.co	(Null)	(Null)	1	0
4	Sandy	21	test4@baomidou.co	(Null)	(Null)	1	0
5	张三	278	zhangsan@gmail.cor	2020-12-17 13:35:01	2020-12-17 13:38:53	1	0
1339437901729087489	张三	20	zhangsan@gmail.cor	(Null)	(Null)	1	0
1339437901729087490	张三	20	zhangsan@gmail.cor	(Null)	(Null)	1	0
1339461202195099649	张三	20	zhangsan@gmail.cor	2020-12-17 06:43:31	2020-12-17 06:43:31	1	0
1339461711110885377	王麻子大爷	99	tianliu@gmail.com	2020-12-17 14:45:32	2020-12-17 16:24:18	4	0

```java
@TableLogic//逻辑删除
private Integer deleted;
```

2. 添加配置

```yaml
global-config:
  db-config:
    logic-delete-field: 1
    logic-not-delete-value: 0
```

3. 测试

```java
//逻辑删除
@Test
public void deletedByLogic(){
    userMapper.deleteById(1L);
    userMapper.selectById(1L);
}
```

```
JDBC Connection [HikariProxyConnection@842588842 wrapping com.mysql.cj.jdbc.ConnectionImpl@2c7db926] will not be managed by Spring
==>  Preparing: UPDATE user SET deleted=1 WHERE id=? AND deleted=0
==> Parameters: 1(Long)
<==    Updates: 1                                    删除只会改delete字段，查询会自动添加delete字段作为条件查询
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@33f2df51]
Creating a new SqlSession
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@642857e0] was not registered for synchronization because synchronization is not active
JDBC Connection [HikariProxyConnection@943501193 wrapping com.mysql.cj.jdbc.ConnectionImpl@2c7db926] will not be managed by Spring
==>  Preparing: SELECT id,name,age,email,gmt_create,gmt_modified,version,deleted FROM user WHERE id=? AND deleted=0
==> Parameters: 1(Long)
<==      Total: 0
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@642857e0]
```

性能分析插件

日常开发中，会遇到慢sql。 通过测试，druid....

mp执行 SQL 分析打印

👉 mybatis-plus-sample-crud(opens new window)

- p6spy 依赖引入

```xml
<!-- p6spy -->
<dependency>
    <groupId>p6spy</groupId>
    <artifactId>p6spy</artifactId>
    <version>3.9.1</version>
</dependency>
```

- 修改配置文件

```
spring:
  datasource:
  #修改
    #driver-class-name: com.mysql.cj.jdbc.Driver
    driver-class-name: com.p6spy.engine.spy.P6SpyDriver
    #mysql8 要写时区
    #url: jdbc:mysql://localhost:3306/mybatis_plus?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEn
coding=utf-8
    url: jdbc:p6spy:mysql://localhost:3306/mybatis_plus?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEn
coding=utf-8
    username: root
    password: root
```

```
#p6spy
#3.2.1以上使用
modulelist=com.baomidou.mybatisplus.extension.p6spy.Myba
tisPlusLogFactory,com.p6spy.engine.outage.P6OutageFactor
y
#3.2.1以下使用或者不配置
#modulelist=com.p6spy.engine.logging.P6LogFactory,com.p6
spy.engine.outage.P6OutageFactory
# 自定义日志打印
logMessageFormat=com.baomidou.mybatisplus.extension.p6sp
y.P6SpyLogger
#日志输出到控制台
appender=com.baomidou.mybatisplus.extension.p6spy.Stdout
Logger
# 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
```

条件构造器

文档： https://baomidou.com/guide/wrapper.html#and

```
@Autowired
private UserMapper userMapper;

@Test
public void test1(){
    QueryWrapper<User> userWrapper = new QueryWrapper<>
();
    userWrapper.eq("name", "张三");
    List<User> users =
userMapper.selectList(userWrapper);
    System.out.println(users);
}

@Test
public void test2(){
    QueryWrapper<User> userQueryWrapper = new
QueryWrapper<>();
```

```
    userQueryWrapper.inSql("id", "select id from user
where id < 3");
    List<User> userList =
userMapper.selectList(userQueryWrapper);
    userList.forEach(System.out::println);
}

@Test
public void test3(){
    QueryWrapper<User> userQueryWrapper = new
QueryWrapper<>();
    userQueryWrapper.and(i->i.eq("name", "张
三").eq("age", 278));
    List<User> userList =
userMapper.selectList(userQueryWrapper);
    userList.forEach(System.out::println);
}
```

代码自动生成器