## 1.目标

1）了解Mybatis-Plus

2）整合Mybatis-Plus

3）通用CRUD Mybatis-Plus的配置

4）条件构造器 Mybatis-Plus 的Service封装

5）代码生成器

## 2.Mybatis-Plus介绍

### 2.1.MyBatis-Plus介绍

MyBatis-Plus（简称 MP）是一个 MyBatis的增强工具，**在 MyBatis 的基础上只做增强不做改变**，为简化开发提高效率而生。该框架由baomidou（苞米豆）组织开发并且开源的。

官网：https://mybatis.plus/ 或 <https://mp.baomidou.com/>

github地址： <https://github.com/baomidou/mybatis-plus>

码云地址：<https://gitee.com/baomidou/mybatis-plus>





### 2.2.支持的数据库

MyBatisPlus支持如下数据库：

mysql mariadb oracle db2 h2 hsql sqlite postgresql sqlserver

达梦数据库 虚谷数据库 人大金仓数据库

### 2.3.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

**支持多种数据库**：支持 MySQL MariaDB Oracle DB2 H2 HSQL SQLite Postgre SQLServer2005 SQLServer 等多种数据库

**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1

**支持XML热加载**：Mapper 对应的 XML 支持热加载，对于简单的 CRUD 操作，甚至可以无XML启动

**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 ）

支持关键词自动转义：支持数据库关键词（order key......）自动转义，还可自定义关键词

**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

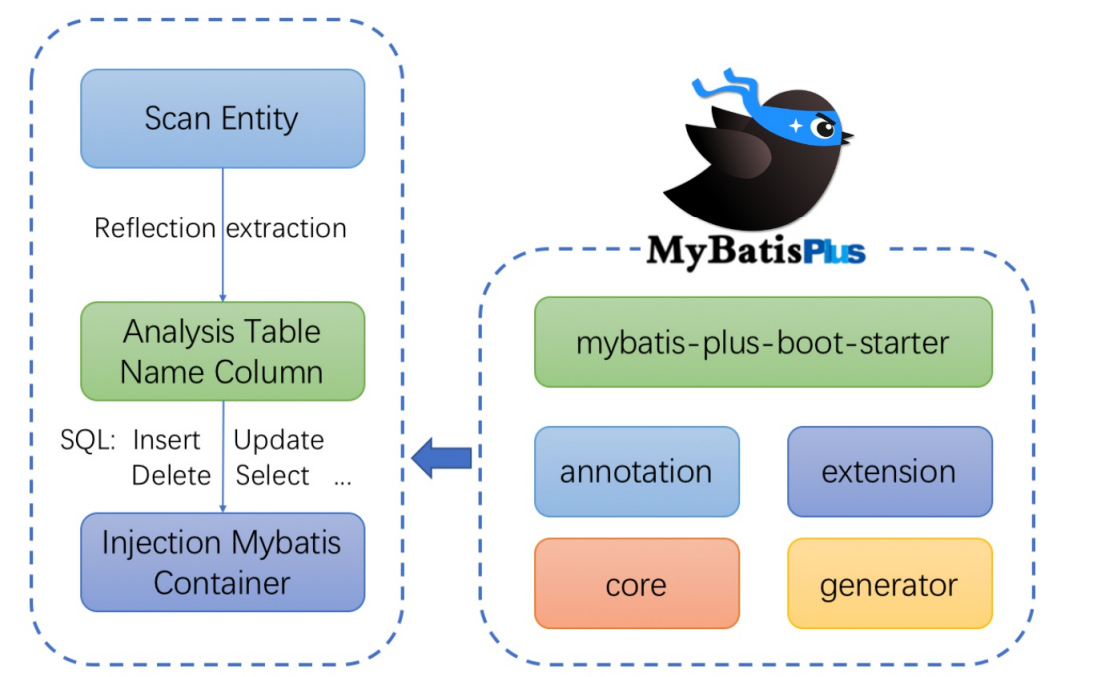
**内置分页插件**：基于 MyBatis 物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同于普通 List 查询

**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

**内置 Sql 注入剥离器**：支持 Sql 注入剥离，有效预防 Sql 注入攻击

### 2.4.架构



Mybatis主要包含以下模块：

**核心功能(core)**，基于Mybatis的封装，提供了Mybatis Plus的基础配置类与核心功能，如内置通用Mapper， Lambda 表达式查询等。

**注解(annotation)**，提供了Mybatis Plus中注解的定义。

**扩展功能(extension)**，提供扩展及插件功能，包括分页插件通用Service扩展性能分析插件等。

**代码生成器(generator)**：通过代码生成器可以快速生成 Mapper接口Entity实体类 Mapper XML Service Controller 等各个模块的代码，极大的提升了开发效率。

执行流程：

（1）扫描注解Entity，反射提取注解信息如：表名称 字段名称等信息。

（2）分析注解信息并基于com.baomidou.mybatisplus.core.enums的SQL模板生成基本CRUD SQL。

（3）最后将这些SQL注入到Mybatis环境中。

因此**Mybatis plus无需编写CRUD SQL语句，只需继承BaseMapper，魔术般的拥有了CRUD功能(通用CRUD)。**

## 3.快速入门

### 3.1.准备环境

JDK 8+

~~Maven 3.3.9~~

Maven 3.6.3

~~IDEA 2018.2~~

Eclipse 2020-03 (4.15.0)

~~MySQL5.7~~

MySQL10.0.11

### 3.2.创建数据库及表

创建数据库并设置字符集为utf-8：

|  |
| --- |
| CREATE DATABASE `mp` CHARACTER SET 'utf8' COLLATE 'utf8\_general\_ci'; |

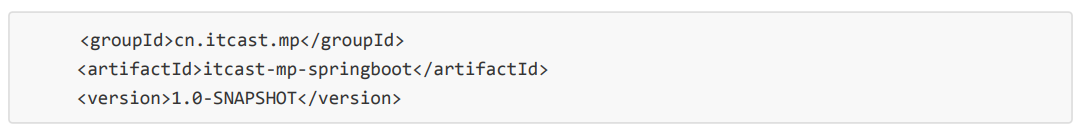
创建表和测试数据：

|  |
| --- |
| ‐‐ 创建测试表  CREATE TABLE `tb\_user` ( `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键ID', `user\_name` varchar(20) NOT NULL COMMENT '用户名', `password` varchar(20) NOT NULL COMMENT '密码', `name` varchar(30) DEFAULT NULL COMMENT '姓名', `age` int(11) DEFAULT NULL COMMENT '年龄', `email` varchar(50) DEFAULT NULL COMMENT '邮箱', `birthday` datetime DEFAULT NULL, PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;  ‐‐ 插入测试数据  insert into `tb\_user`(`id`,`user\_name`,`password`,`name`,`age`,`email`,`birthday`) values (1,'zhangsan','123456','张三',18,'test1@itcast.cn','2019‐09‐26 11:42:01'), (2,'lisi','123456','李四',20,'test2@itcast.cn','2019‐10‐01 11:42:08'), (3,'wangwu','123456','王五',28,'test3@itcast.cn','2019‐10‐02 11:42:14'), (4,'zhaoliu','123456','赵六',21,'test4@itcast.cn','2019‐10‐05 11:42:18'), (5,'sunqi','123456','孙七',24,'test5@itcast.cn','2019‐10‐14 11:42:23'); |

### 3.3.工程搭建

#### 3.3.1.创建工程

创建maven工程，分别填写GroupId ArtifactId和Version，如下：



#### 3.3.2.导入依赖

导入maven依赖，由于本例采用Spring boot技术，使用mybatis-plus-boot-starter能与其便捷集成：

版本规划： Spring boot：

2.1.3.RELEASE

mybatis-plus： 3.1.0

|  |
| --- |
| <?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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itcast.mp</groupId>  <artifactId>itcast-mp-springboot</artifactId>  <version>1.0-SNAPSHOT</version>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.3.RELEASE</version>  </parent>  <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>  **<!--简化代码的工具包-->**  **<dependency>**  **<groupId>org.projectlombok</groupId>**  **<artifactId>lombok</artifactId>**  **<optional>true</optional>**  **</dependency>**  **<!--mybatis-plus的springboot支持-->**  **<dependency>**  **<groupId>com.baomidou</groupId>**  **<artifactId>mybatis-plus-boot-starter</artifactId>**  **<version>3.1.0</version>**  **</dependency>**  **<!--mysql驱动-->**  **<dependency>**  **<groupId>mysql</groupId>**  **<artifactId>mysql-connector-java</artifactId>**  **<version>8.0.11</version>**  **</dependency>**  </dependencies>  </project> |

#### 3.3.3.编写application.properties

|  |
| --- |
| **spring.application.name = itcast-mp-springboot**  **spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver**  **spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mp?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&useSSL=false**  **spring.datasource.username=root**  **spring.datasource.password=mysql**  #指定mybatis-config.xml的位置  mybatis-plus.config-location = classpath:mybatis/mybatis-config.xml  #指定mapper文件位置  mybatis-plus.mapper-locations = classpath\*:mybatis/mapper/\*.xml  #别名配置  mybatis-plus.type-aliases-package = cn.itcast.mp.pojo  #开启自动驼峰映射，注意：配置configuration.map-underscore-to-camel-case则不能配置config-location  #mybatis-plus.configuration.map-underscore-to-camel-case=true  **# Logger Config**  **logging.level.root: debug** |

#### 3.3.4.编写pojo

|  |
| --- |
| package cn.itcast.mp.pojo;  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;  import lombok.NoArgsConstructor;  import lombok.ToString;  import java.time.LocalDateTime;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @Data  @ToString  @NoArgsConstructor  @AllArgsConstructor  @TableName("tb\_user")  public class User {  @TableId(value = "id",type= IdType.ID\_WORKER)//指定主键的字段名  private Long id;  @TableField("user\_name")//指定对应字段名  private String userName;  private String password;  @TableField("NAME")  private String name;  @TableField("AGE")  private Integer age;  @TableField("EMAIL")  private String email;  @TableField("birthday")  private LocalDateTime birthday;  } |

@Data：lombok的注解，使用它可以省略getter/setter方法。

@NoArgsConstructor： 生成无参构造方法。

@AllArgsConstructor：生成所有参数构造方法，参数顺序与属性定义顺序一致。 @TableName：指定表名。

@TableId：指定主键名。

@TableField：指定列名。

#### 3.3.5.编写mapper

|  |
| --- |
| package cn.itcast.mp.mapper;  import cn.itcast.mp.pojo.User;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  /\*\*  \* Created by Administrator.  \*/  public interface UserMapper **extends BaseMapper<User>** {  public User findById(Long id);  } |

#### 3.3.6.编写启动类

|  |
| --- |
| package cn.itcast.mp;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @SpringBootApplication  @MapperScan("cn.itcast.mp.mapper")  public class MyApplication {  public static void main(String[] args) {  SpringApplication.run(MyApplication.class,args);  }  } |

#### 3.3.7.编写测试用例

编写UserMapper的测试用例 ，使用UserMapper查询用户列表。

在test下创建测试类，包名为 cn.itcast.mp。

|  |
| --- |
| package cn.itcast.mp;  import cn.itcast.mp.mapper.UserMapper;  import cn.itcast.mp.pojo.User;  import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  import java.time.LocalDateTime;  import java.time.format.DateTimeFormatter;  import java.util.Arrays;  import java.util.List;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @RunWith(SpringRunner.class)  @SpringBootTest  public class UserMapperTest {  @Autowired  UserMapper userMapper;  **@Test**  **public void testSelect(){**  **//查询tb\_user记录**  **List<User> users = userMapper.selectList(null);**  **System.out.println(users);**  **}**  @Test  public void testFindByid(){  //查询tb\_user记录  User user = userMapper.findById(2L);  System.out.println(user);  }  //插入  @Test  public void testInsert(){  User user = new User();  // user.setId(100L);  user.setName("曹操");  user.setPassword("111111");  user.setAge(20);  user.setUserName("caocao");  DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-DD HH:mm:ss");  LocalDateTime localDateTime = LocalDateTime.parse("1990-01-01 00:00:00", dateTimeFormatter);  user.setBirthday(localDateTime);  int i = userMapper.insert(user);  System.out.println(i);  }  //==================================================  @Test  public void testUpdate(){  User user = new User();  //更新记录的主键值  user.setId(100L);  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.updateById(user);  System.out.println(i);  }  //根据条件进行更新  @Test  public void testUpdate2(){  User user = new User();  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  //设置条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","曹操");  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.update(user,queryWrapper);  System.out.println(i);  }  //根据条件进行更新，可以将为NULl的值更新到数据库  @Test  public void testUpdate3(){  UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();  updateWrapper.eq("name","曹操").set("birthday",null);  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.update(null,updateWrapper);  System.out.println(i);  }  //==========================================  @Test  public void testDelete(){  //设置条件  // QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("age",100);  // queryWrapper.eq("name","曹操1");  User user = new User();  user.setAge(100);  user.setName("曹操1");  QueryWrapper<User> queryWrapper =new QueryWrapper<>(user);  //根据条件删除  int delete = userMapper.delete(queryWrapper);  System.out.println(delete);  }  //批量删除  @Test  public void testDelete2(){  //根据条件删除  int delete = userMapper.deleteBatchIds(Arrays.asList(101L,102L,103L));  System.out.println(delete);  }  //==================================  @Test  public void testSelectOne(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","张三");  //根据条件查询，只能查询出一条记录，否则报错  User user = userMapper.selectOne(queryWrapper);  System.out.println(user);  }  @Test  public void testSelectCount(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //根据条件查询，只能查询出一条记录，否则报错  Integer count = userMapper.selectCount(queryWrapper);  System.out.println(count);  }  @Test  public void testSelectPage(){  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //用构造方法设置当前页码，每页记录数  int pageIndex = 2;//当前页码  int size = 3;//每页记录数  Page<User> page =new Page<>(pageIndex,size);  IPage<User> userIPage = userMapper.selectPage(page, queryWrapper);  long pages = userIPage.getPages();//总页数  long total = userIPage.getTotal();//总记录数  //记录列表  List<User> records = userIPage.getRecords();  System.out.println(records);  }  //===========================================  @Test  public void testEq() {  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","曹操");//等于  queryWrapper.gt("age",20);//大于?  queryWrapper.in("user\_name","caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testEq2() {  //条件  LambdaQueryWrapper<User> queryWrapper =new LambdaQueryWrapper<>();  queryWrapper.eq(User::getName,"曹操");//等于  queryWrapper.gt(User::getAge,20);//大于?  queryWrapper.in(User::getUserName,"caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testEq3() {  //条件  LambdaQueryWrapper<User> queryWrapper =new LambdaQueryWrapper<>();  String name = null;  Integer age = 20;  queryWrapper.eq(name!=null && !name.equals(""),User::getName,name);//等于  queryWrapper.gt(age!=null,User::getAge,age);//大于?  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testWrapper() {  QueryWrapper<User> wrapper = new QueryWrapper<>();  //SELECT id,user\_name,password,name,age,email FROM tb\_user WHERE name LIKE ?  //Parameters: %曹%(String)  wrapper.likeRight("name", "曹");  wrapper.select("id","name");  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.out.println(user);  }  }  } |

执行查看测试结果。

## 4.常见配置

在MP中有大量的配置，其中有一部分是Mybatis原生的配置，另一部分是MP的配置，详情： <https://mybatis.plus/config/>

下面我们对常用的配置做讲解。

### 4.1.configLocations

configLocations即MyBatis 配置文件位置，如果您有单独的MyBatis 配置，请将其路径配置到 configLocation 中。 MyBatis Configuration 的具体内容请参考MyBatis 官方文档

示例：

1）在resources下创建mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <settings>  <!--mapUnderscoreToCamelCase=true-->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings>  <!--<plugins>-->  <!--<plugin interceptor="com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor"></plugin>-->  <!--</plugins>-->  </configuration> |

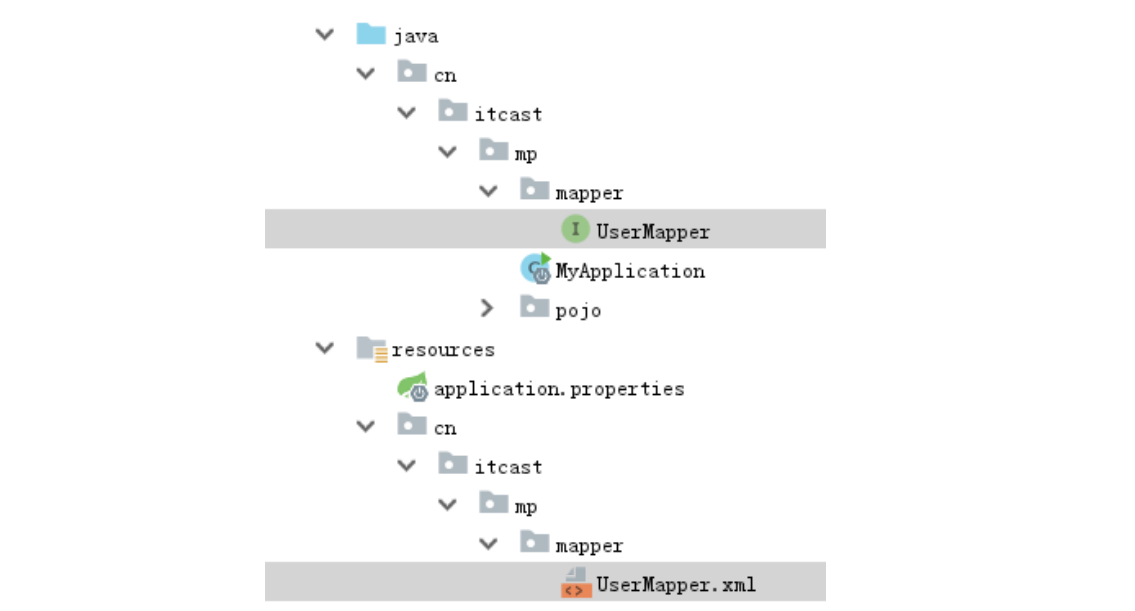
2）在application.properties下配置configLocations，如下：

|  |
| --- |
| mybatis‐plus.config‐location = classpath:mybatis‐config.xml |

### 4.2.mapperLocations

mapperLocations即MyBatis Mapper 所对应的mapper配置文件位置，如果您在 Mapper 中有自定义方法 （XML 中有自定义实现），需要进行该配置，告诉 Mapper所对应的 XML 文件位置。

如果不配置mapperLocations时，mapper的xml文件存放路径需要和mapper class文件保持一致，文件名保持 一 致，如下：



测试：

新建UserMapper.xml：

将此文件放在resources/cn/itcast/mp/mapper下

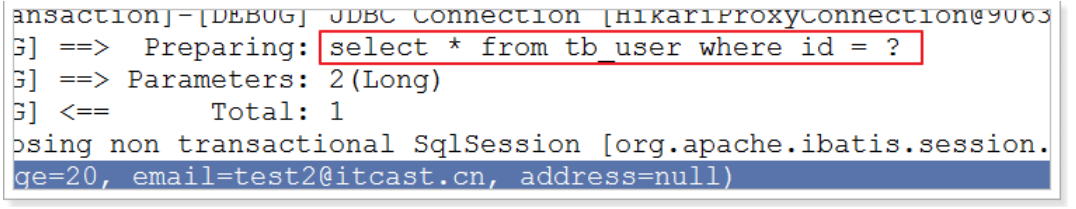
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="cn.itcast.mp.mapper.UserMapper">  <select id="findById" resultType="user" parameterType="java.lang.Long">  select \* from tb\_user where id = #{id}  </select>  </mapper> |

|  |
| --- |
| package cn.itcast.mp.mapper;  import cn.itcast.mp.pojo.User;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  /\*\*  \* Created by Administrator.  \*/  public interface UserMapper extends BaseMapper<User> {  public User findById(Long id);  } |

测试用例：

|  |
| --- |
| package cn.itcast.mp;  import cn.itcast.mp.mapper.UserMapper;  import cn.itcast.mp.pojo.User;  import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  import java.time.LocalDateTime;  import java.time.format.DateTimeFormatter;  import java.util.Arrays;  import java.util.List;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @RunWith(SpringRunner.class)  @SpringBootTest  public class UserMapperTest {  @Autowired  UserMapper userMapper;  @Test  public void testSelect(){  //查询tb\_user记录  List<User> users = userMapper.selectList(null);  System.out.println(users);  }  **@Test**  **public void testFindByid(){**  **//查询tb\_user记录**  **User user = userMapper.findById(2L);**  **System.out.println(user);**  **}**  //插入  @Test  public void testInsert(){  User user = new User();  // user.setId(100L);  user.setName("曹操");  user.setPassword("111111");  user.setAge(20);  user.setUserName("caocao");  DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-DD HH:mm:ss");  LocalDateTime localDateTime = LocalDateTime.parse("1990-01-01 00:00:00", dateTimeFormatter);  user.setBirthday(localDateTime);  int i = userMapper.insert(user);  System.out.println(i);  }  //==================================================  @Test  public void testUpdate(){  User user = new User();  //更新记录的主键值  user.setId(100L);  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.updateById(user);  System.out.println(i);  }  //根据条件进行更新  @Test  public void testUpdate2(){  User user = new User();  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  //设置条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","曹操");  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.update(user,queryWrapper);  System.out.println(i);  }  //根据条件进行更新，可以将为NULl的值更新到数据库  @Test  public void testUpdate3(){  UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();  updateWrapper.eq("name","曹操").set("birthday",null);  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.update(null,updateWrapper);  System.out.println(i);  }  //==========================================  @Test  public void testDelete(){  //设置条件  // QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("age",100);  // queryWrapper.eq("name","曹操1");  User user = new User();  user.setAge(100);  user.setName("曹操1");  QueryWrapper<User> queryWrapper =new QueryWrapper<>(user);  //根据条件删除  int delete = userMapper.delete(queryWrapper);  System.out.println(delete);  }  //批量删除  @Test  public void testDelete2(){  //根据条件删除  int delete = userMapper.deleteBatchIds(Arrays.asList(101L,102L,103L));  System.out.println(delete);  }  //==================================  @Test  public void testSelectOne(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","张三");  //根据条件查询，只能查询出一条记录，否则报错  User user = userMapper.selectOne(queryWrapper);  System.out.println(user);  }  @Test  public void testSelectCount(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //根据条件查询，只能查询出一条记录，否则报错  Integer count = userMapper.selectCount(queryWrapper);  System.out.println(count);  }  @Test  public void testSelectPage(){  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //用构造方法设置当前页码，每页记录数  int pageIndex = 2;//当前页码  int size = 3;//每页记录数  Page<User> page =new Page<>(pageIndex,size);  IPage<User> userIPage = userMapper.selectPage(page, queryWrapper);  long pages = userIPage.getPages();//总页数  long total = userIPage.getTotal();//总记录数  //记录列表  List<User> records = userIPage.getRecords();  System.out.println(records);  }  //===========================================  @Test  public void testEq() {  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","曹操");//等于  queryWrapper.gt("age",20);//大于?  queryWrapper.in("user\_name","caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testEq2() {  //条件  LambdaQueryWrapper<User> queryWrapper =new LambdaQueryWrapper<>();  queryWrapper.eq(User::getName,"曹操");//等于  queryWrapper.gt(User::getAge,20);//大于?  queryWrapper.in(User::getUserName,"caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testEq3() {  //条件  LambdaQueryWrapper<User> queryWrapper =new LambdaQueryWrapper<>();  String name = null;  Integer age = 20;  queryWrapper.eq(name!=null && !name.equals(""),User::getName,name);//等于  queryWrapper.gt(age!=null,User::getAge,age);//大于?  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  }  @Test  public void testWrapper() {  QueryWrapper<User> wrapper = new QueryWrapper<>();  //SELECT id,user\_name,password,name,age,email FROM tb\_user WHERE name LIKE ?  //Parameters: %曹%(String)  wrapper.likeRight("name", "曹");  wrapper.select("id","name");  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.out.println(user);  }  }  } |

运行结果：



也可以给mapper XML文件定义一个和mapper接口不同的存放路径 ，如下：

|  |
| --- |
| mybatis‐plus.mapper‐locations = classpath\*:mybatis/mapper/\*.xml |

**Maven 多模块项目的扫描路径需以 classpath\*: 开头 （即加载多个 jar 包下的 XML 文件）。**

### 4.3.typeAliasesPackage

设置MyBaits 别名包扫描路径，通过该属性可以给包中的类注册别名，注册后在 Mapper 对应的 XML 文件中可以 直接使用类名，而不用使用全限定的类名（即 XML 中调用的时候不用包含包名）。

示例：

|  |
| --- |
| mybatis‐plus.type‐aliases‐package = cn.itcast.mp.pojo |

### 4.4.mapUnderscoreToCamelCase

**类型： boolean 默认值： true**

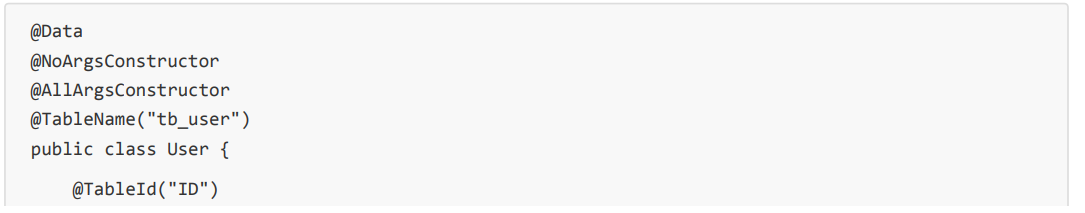
是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A\_COLUMN（下划线命名） 到经典Java 属性名aColumn（驼峰命名） 的类似映射。

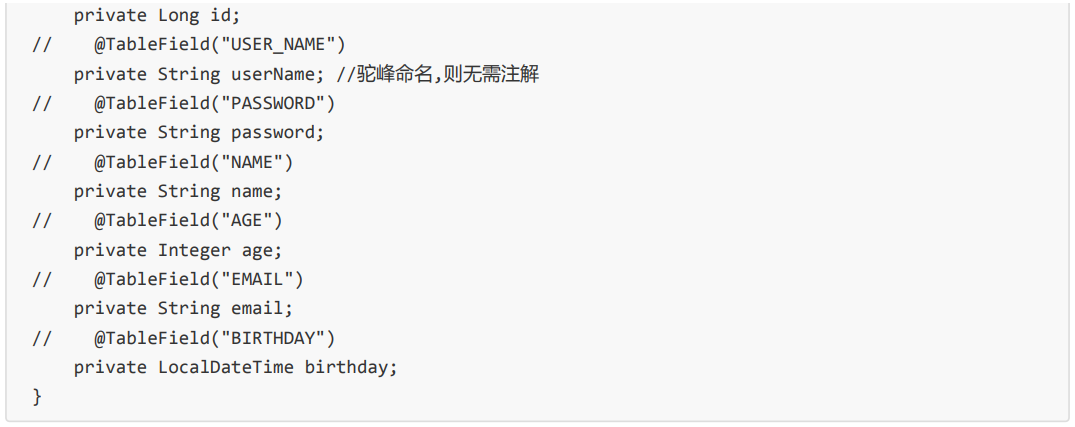
注意： 在 MyBatis-Plus 中此属性默认值为true，用于生成最终的 SQL 语句。如果您的数据库命名符合规则无需使用 @TableField 注解指定数据库字段名。

测试：

|  |
| --- |
| #开启自动驼峰映射，注意：配置configuration.map‐underscore‐to‐camel‐case则不能配置config‐location mybatis‐plus.configuration.map‐underscore‐to‐camel‐case=true |

1. 屏蔽@TableField





2）测试userMapper.findById方法

跟踪发现userName可以映射成功。



如果项目中有符合驼峰规则的定义也有不符合的，此时建议统一使用@TableField。

3）如果使用mybatis-config.xml的同时在application.properties配置mybatis-plus.configuration则报错。

Property 'configuration' and 'configLocation' can not specified with together

解决方法：

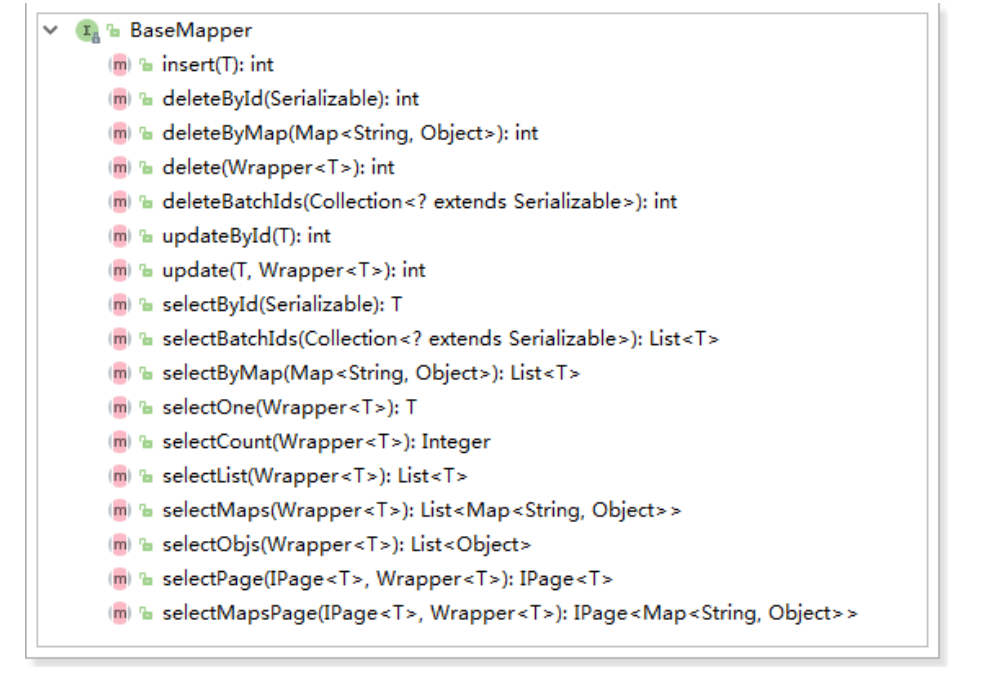
只使用一种配置方法。

本案例屏蔽mybatis-plus.configuration.map-underscore-to-camel-case=true，在mybatis-config.xml中配置 settings。

|  |
| --- |
| <settings>  <!--mapUnderscoreToCamelCase=true-->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings> |

## 5.通用CRUD

通过前面的学习，我们了解到通过继承BaseMapper就可以获取到各种各样的单表操作，接下来我们将详细讲解这 些操作，下图是BaseMapper的各个方法：



### 5.1.插入操作

#### 5.1.1.方法定义

|  |
| --- |
| /\*\*  \* <p>  \* 插入一条记录  \* </p>  \*  \* **@param** entity 实体对象  \*/  **int** insert(T entity); |

#### 5.1.2.测试用例

|  |
| --- |
| //插入  @Test  public void testInsert(){  User user = new User();  // user.setId(100L);  user.setName("曹操");  user.setPassword("111111");  user.setAge(20);  user.setUserName("caocao");  DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-DD HH:mm:ss");  LocalDateTime localDateTime = LocalDateTime.parse("1990-01-01 00:00:00", dateTimeFormatter);  user.setBirthday(localDateTime);  **int i = userMapper.insert(user); // 返回的i是受影响的行数，并不是自增后的id**  System.out.println(i);  } |

#### 5.1.3.MP主键生成策略

上例中Mybatis-plus自动生成ID，如何设置id的生成策略呢？

MP支持的id策略如下：

|  |
| --- |
| /\*  \* Copyright (c) 2011-2020, hubin (jobob@qq.com).  \* <p>  \* Licensed under the Apache License, Version 2.0 (the "License");  \* you may not use this file except in compliance with the License.  \* You may obtain a copy of the License at  \* <p>  \* https://www.apache.org/licenses/LICENSE-2.0  \* <p>  \* Unless required by applicable law or agreed to in writing, software  \* distributed under the License is distributed on an "AS IS" BASIS,  \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  \* See the License for the specific language governing permissions and  \* limitations under the License.  \*/  package com.baomidou.mybatisplus.annotation;  import lombok.Getter;  /\*\*  \* 生成ID类型枚举类  \*  \* @author hubin  \* @since 2015-11-10  \*/  @Getter  public enum IdType {  **/\*\***  **\* 数据库ID自增**  **\*/**  **AUTO(0),**  **/\*\***  **\* 该类型为未设置主键类型**  **\*/**  **NONE(1),**  **/\*\***  **\* 用户输入ID**  **\* <p>该类型可以通过自己注册自动填充插件进行填充</p>**  **\*/**  **INPUT(2),**  **/\* 以下3种类型、只有当插入对象ID 为空，才自动填充。 \*/**  **/\*\***  **\* 全局唯一ID (idWorker)**  **\*/**  **ID\_WORKER(3),**  **/\*\***  **\* 全局唯一ID (UUID)**  **\*/**  **UUID(4),**  **/\*\***  **\* 字符串全局唯一ID (idWorker 的字符串表示)**  **\*/**  **ID\_WORKER\_STR(5);**  private final int key;  IdType(int key) {  this.key = key;  }  } |

1. **自增主键**

完全采用数据库自增主键方式。

A.设置mysql数据库主键为自增

B.修改User对象

|  |
| --- |
| @TableId(value = "ID", type = IdType.AUTO)  private Long id;  或:  @TableId(value = "ID")  private Long id; |

**C.程序中不用设置主键**

1. 输入主键

手动设置主键值。

A.mysql数据库主键为自增或不是自增都可以

B.修改User对象

|  |
| --- |
| @TableId(value = "ID",type = IdType.INPUT)  private Long id; |

C.程序中需要设置主键

1. UUID

生成全局唯一ID。

A.mysql数据库主键为字符串类型，不是自增类型。

B.修改User对象。

|  |
| --- |
| @TableId(value = "ID",type = IdType.UUID)  private String id; |

C.程序中不用设置主键

1. ID\_WORKER\_STR

采用雪花片算法（**雪花算法生成的ID是纯数字且具有时间顺序，适合分布式场景**）生成全局唯一ID，**字符串类型**。

A.mysql数据库主键为字符串类型，不是自增类型。

B.修改User对象。

|  |
| --- |
| @TableId(value = "ID",type = IdType.ID\_WORKER\_STR)  private String id; |

C.程序中不用设置主键

5）ID\_WORKER

**采用雪花片算法生成全局唯一ID，数值类型。**

A.mysql数据库主键为数值类型，不是自增类型。

B.修改User对象。

|  |
| --- |
| @TableId(value = "ID",type = IdType.ID\_WORKER)  private Long id; |

C.程序中不用设置主键

### 5.2.更新操作

#### 5.2.1.根据id更新

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 ID 修改  \* </p>  \*  \* **@param** entity 实体对象  \*/  **int** updateById(@Param(Constants.***ENTITY***) T entity); |

根据id更新操作步骤：

1 首先需要设置对象的主键属性值。

2 再设置要更新的属性值。

3 根据主键找到对象，更新设置属性值。

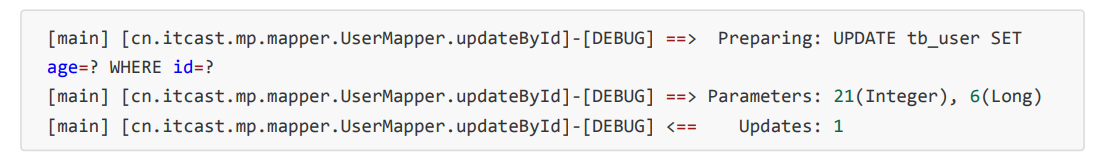
4 返回影响的记录数。

**注意：只能将对象中不为NULL的属性更新到表中。**

测试：

|  |
| --- |
| @Test  public void testUpdate(){  User user = new User();  //更新记录的主键值  user.setId(100L);  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.updateById(user);  System.out.println(i);  } |

结果：



#### 5.2.2.根据条件更新

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 whereEntity 条件，更新记录  \* </p>  \*  \* **@param** entity 实体对象 (set 条件值,不能为 null)  \* **@param** 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）  \*/  **int** update(@Param(Constants.***ENTITY***) T entity, @Param(Constants.***WRAPPER***) Wrapper<T> updateWrapper); |

根据ID更新一次只能更新一条记录，根据条件更新可实现批量更新。

根据条件更新步骤：

1 在对象中设置要更新的属性值。

2 设置QueryWrapper，设置更新条件，可以设置多个。

3 返回影响的记录数。

**注意：只能将对象中不为NULL的属性更新到表中。**

测试用例： 下次将name等于“曹操”的记录全部更新。

|  |
| --- |
| //根据条件进行更新  @Test  public void testUpdate2(){  User user = new User();  user.setAge(100);//要更新的值  user.setPassword("12222");//要更新的值  **//设置条件**  **QueryWrapper<User> queryWrapper =new QueryWrapper<>();**  **queryWrapper.eq("name","曹操");**  //只将对象中不为NULL的值更新到数据库中  int i = userMapper.update(user,queryWrapper);  System.out.println(i);  } |

### 5.3.删除操作

#### 5.3.1.根据ID删除

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 ID 删除  \* </p>  \*  \* **@param** id 主键ID  \*/  **int** deleteById(Serializable id); |

操作步骤：

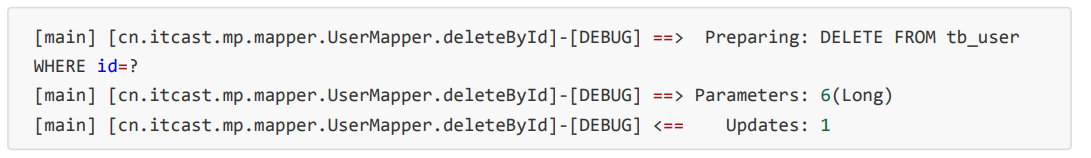
1 指定要删除记录的主键值。

2 调用deleteById方法执行删除。

测试用例：

|  |
| --- |
| @Test  public void testDeleteById() {  //执行删除操作  int result = **this.userMapper.deleteById(6L);**  System.out.println("result = " + result);  } |

结果：



#### 5.3.2.根据条件删除

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 entity 条件，删除记录  \* </p>  \*  \* **@param** queryWrapper 实体对象封装操作类（可以为 null）  \*/  **int** delete(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper); |

根据条件删除步骤：

1 定义对象，设置属性值，指定删除条件 ，可指定多个删除条件

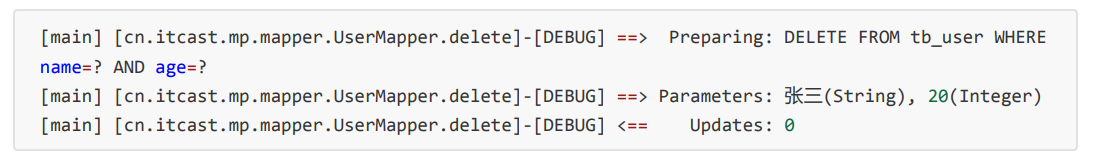
**注意：删除条件只匹配对象中不为NULL的属性值**

2 设置QueryWrapper

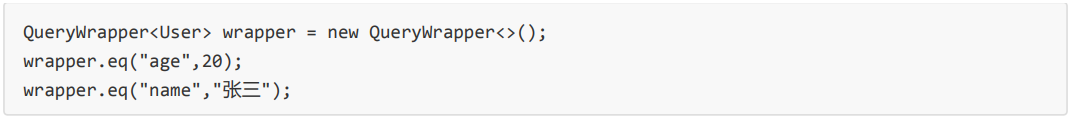
3 执行删除

|  |
| --- |
| @Test  public void testDelete(){  //设置条件  // QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("age",100);  // queryWrapper.eq("name","曹操1");  User user = new User();  user.setAge(100);  user.setName("曹操1");  QueryWrapper<User> queryWrapper =new QueryWrapper<>(user);  //根据条件删除  int delete = userMapper.delete(queryWrapper);  System.out.println(delete);  } |

结果：



**注意： 定义QueryWrapper可以不包装模型对象，手动设置条件**，如下：



#### 5.3.3.根据ID 批量删除

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 删除（根据ID 批量删除）  \* </p>  \*  \* **@param** idList 主键ID列表(不能为 null 以及 empty)  \*/  **int** deleteBatchIds(@Param(Constants.***COLLECTION***) Collection<? **extends** Serializable> idList); |

批量删除操作步骤：

1 指定 id列表

2 执行删除

测试用例：

|  |
| --- |
| //批量删除  @Test  public void testDelete2(){  //根据条件删除  int delete = userMapper.deleteBatchIds(Arrays.asList(101L,102L,103L));  System.out.println(delete);  } |

结果：



### 5.4.查询操作

MP提供了多种查询操作，包括**根据id查询，批量查询，查询单条数据，查询列表，分页查询**等操作。

#### 5.4.1.根据 ID 查询

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 ID 查询  \* </p>  \*  \* **@param** id 主键ID  \*/  T selectById(Serializable id); |

根据id查询步骤：

1 设置查询记录的主键值。

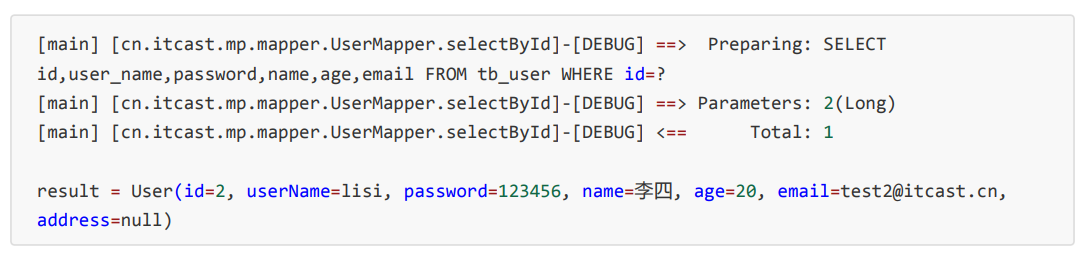
2 执行查询。

3 查询结果返回一个对象。

测试用例：

|  |
| --- |
| User user = **this.userMapper.selectById(2L)**;  System.out.println("result = " + user); |

结果：



#### 5.4.2.查询（根据ID 批量查询）

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 查询（根据ID 批量查询）  \* </p>  \*  \* **@param** idList 主键ID列表(不能为 null 以及 empty)  \*/  List<T> selectBatchIds(@Param(Constants.***COLLECTION***) Collection<? **extends** Serializable> idList); |

根据id列表查询：

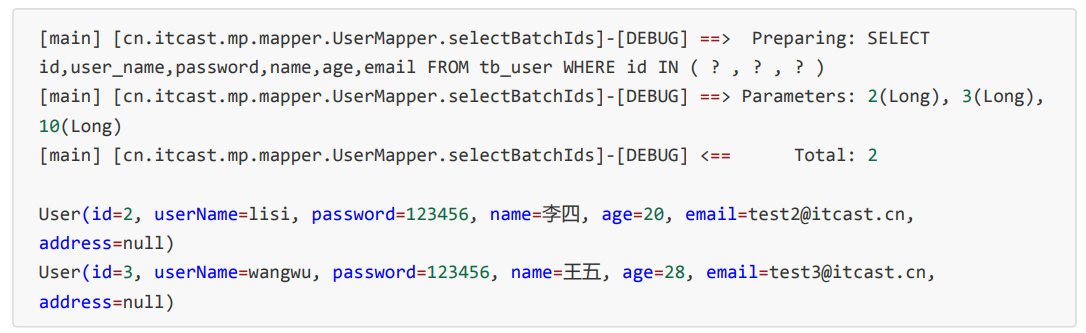
1 设置id列表

2 执行查询

3 查询对象返回List 测试用例：

|  |
| --- |
| @Test  public void testSelectBatchIds() {  //根据id集合批量查询  List users = this.userMapper.selectBatchIds(Arrays.asList(2L, 3L, 10L));  for (User user : users) { System.out.println(user);  }  } |

结果：



#### 5.4.3.根据 entity 条件，查询一条记录

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 entity 条件，查询一条记录  \* </p>  \*  \* **@param** queryWrapper 实体对象  \*/  T selectOne(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper); |

查询步骤：

1 设置QueryWrapper对象，设置查询条件，可以设置多个条件

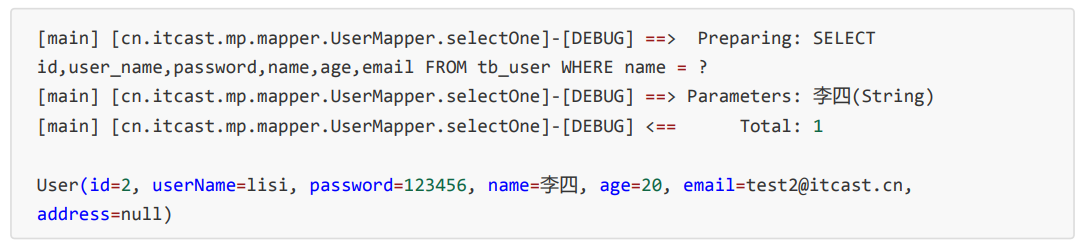
2 执行查询

**注意：如果查询结果为多条记录则报错（TooManyResultsException）。**

测试用例：

|  |
| --- |
| @Test  public void testSelectOne(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","张三");  //根据条件查询，只能查询出一条记录，否则报错  User user = userMapper.selectOne(queryWrapper);  System.out.println(user);  } |

结果：



#### 5.4.4.根据 Wrapper 条件，查询总记录数

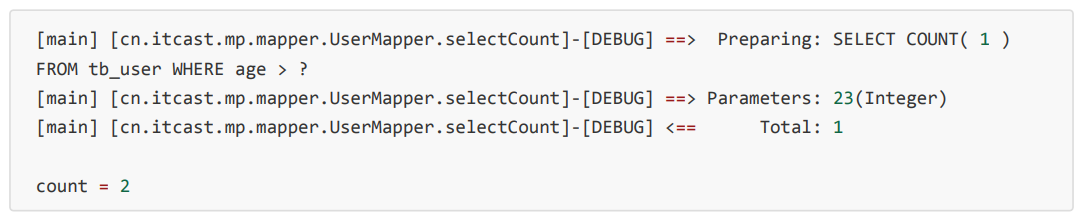
方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 Wrapper 条件，查询总记录数  \* </p>  \*  \* **@param** queryWrapper 实体对象  \*/  Integer selectCount(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper); |

测试用例：

|  |
| --- |
| @Test  public void testSelectCount(){  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //根据条件查询，只能查询出一条记录，否则报错  Integer count = userMapper.selectCount(queryWrapper);  System.out.println(count);  } |

结果：



#### 5.4.5.根据 entity 条件，查询全部记录

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 entity 条件，查询全部记录  \* </p>  \*  \* **@param** queryWrapper 实体对象封装操作类（可以为 null）  \*/  List<T> selectList(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper); |

测试用例：

|  |
| --- |
| @Test  public void testSelectList() {  QueryWrapper wrapper = new QueryWrapper();  wrapper.gt("age", 23);  //年龄大于23岁  //根据条件查询数据  List users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.out.println("user = " + user);  }  } |

#### 5.4.6.根据 entity 条件，查询全部记录（并翻页）

方法定义：

|  |
| --- |
| /\*\*  \* <p>  \* 根据 entity 条件，查询全部记录（并翻页）  \* </p>  \*  \* **@param** page 分页查询条件（可以为 RowBounds.DEFAULT）  \* **@param** queryWrapper 实体对象封装操作类（可以为 null）  \*/  IPage<T> selectPage(IPage<T> page, @Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper); |

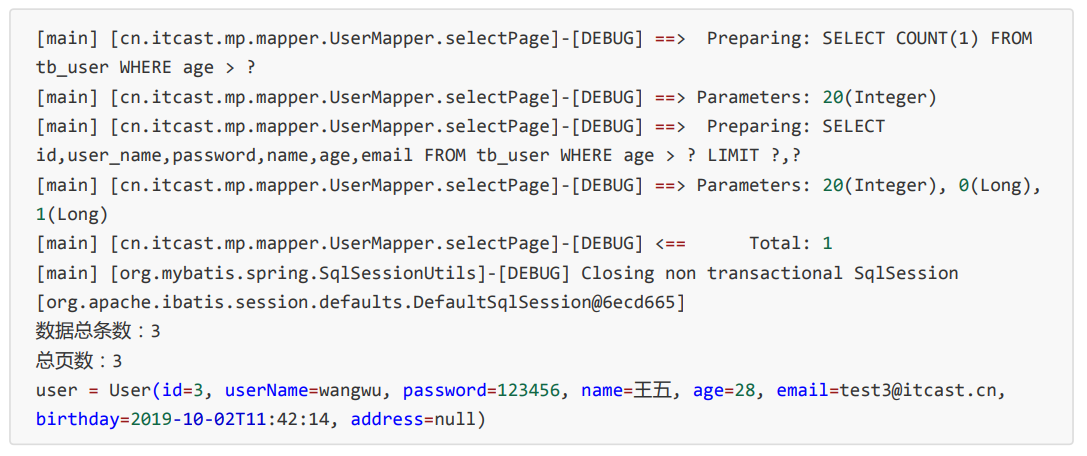
配置分页插件：

|  |
| --- |
| package cn.itcast.mp.config;  import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @Configuration  @MapperScan("cn.itcast.mp.mapper")  public class MybatisPlusConfig {  //配置分页插件的bean  @Bean  public PaginationInterceptor paginationInterceptor(){  return new PaginationInterceptor();  }  } |

测试用例：

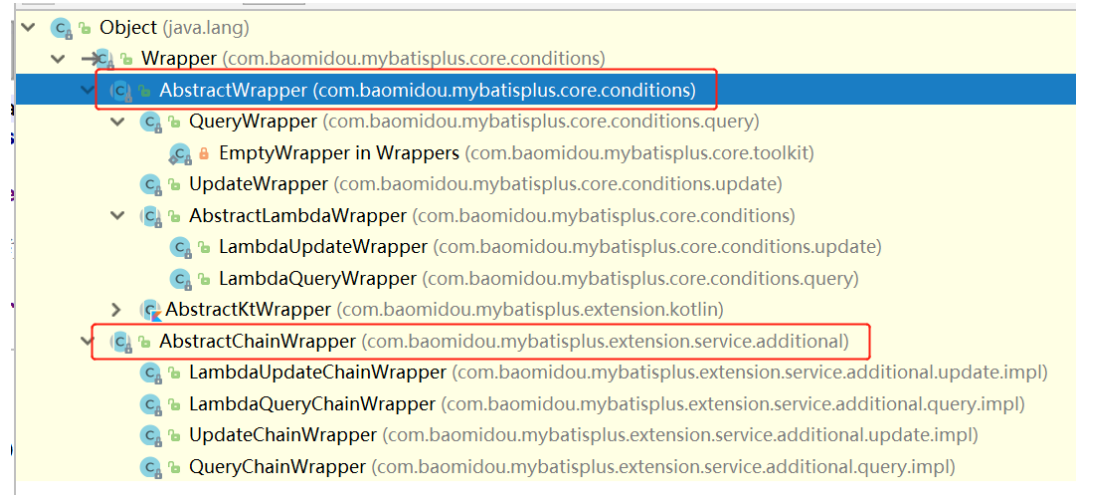
|  |
| --- |
| @Test  public void testSelectPage(){  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  // queryWrapper.eq("name","张三");//等于  queryWrapper.gt("age",20);//大于?  //用构造方法设置当前页码，每页记录数  int pageIndex = 2;//当前页码  int size = 3;//每页记录数  Page<User> page =new Page<>(pageIndex,size);  IPage<User> userIPage = userMapper.selectPage(page, queryWrapper);  long pages = userIPage.getPages();//总页数  long total = userIPage.getTotal();//总记录数  //记录列表  List<User> records = userIPage.getRecords();  System.out.println(records);  } |

结果：



## 6.条件构造器

在MP中，Wrapper接口的实现类关系如下：



**在MP查询中，还可以使用lambda方式查询，降低数据库列表写错的风险。**

### 6.1.基本比较操作

1）eq 等于 =

2）ne 不等于 <>

3）gt 大于 >

4）ge 大于等于 >=

5）lt 小于 <

6）le 小于等于 <=

7）between BETWEEN 值1 AND 值2

8）notBetween NOT BETWEEN 值1 AND 值2

9）in 字段 IN (value.get(0), value.get(1), ...)

10）notIn 字段 NOT IN (v0, v1, ...)

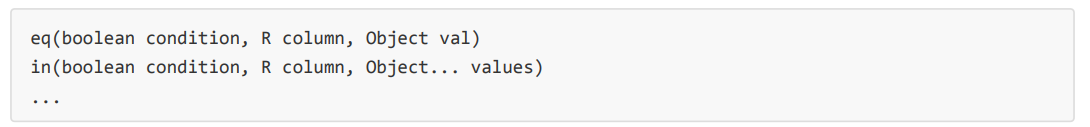
测试用例：

|  |
| --- |
| @Test  public void testEq() {  //条件  QueryWrapper<User> queryWrapper =new QueryWrapper<>();  queryWrapper.eq("name","曹操");//等于  queryWrapper.gt("age",20);//大于?  queryWrapper.in("user\_name","caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  } |

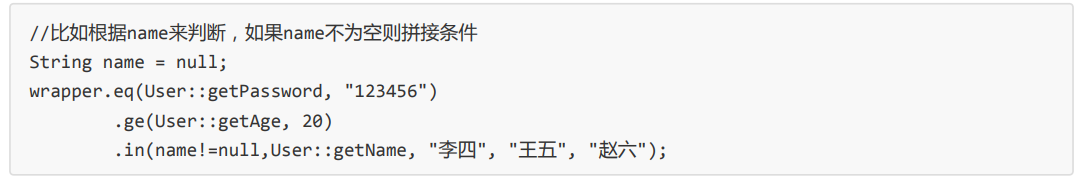
Lambda方式构造条件：

|  |
| --- |
| @Test  public void testEq2() {  //条件  LambdaQueryWrapper<User> queryWrapper =new LambdaQueryWrapper<>();  queryWrapper.eq(User::getName,"曹操");//等于  queryWrapper.gt(User::getAge,20);//大于?  queryWrapper.in(User::getUserName,"caocao");  List<User> users = userMapper.selectList(queryWrapper);  System.out.println(users);  } |

通常在开发中要根据表达式进行判断，表达式为true则拼接条件，如下：



一个例子：



### 6.2.模糊查询

1）like LIKE '%值%'

例: like("name", "王") ---> name like '%王%'

2）notLike NOT LIKE '%值%'

例: notLike("name", "王") ---> name not like '%王%'

3）likeLeft LIKE '%值'

例: likeLeft("name", "王") ---> name like '%王'

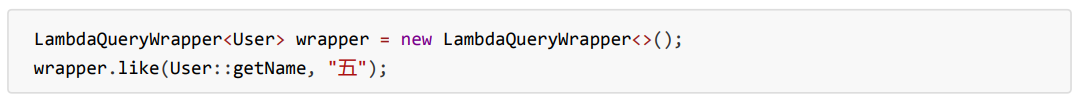
4）likeRight LIKE '值%'

例: likeRight("name", "王") ---> name like '王%'

测试用例：

|  |
| --- |
| @Test  public void testWrapper() {  QueryWrapper<User> wrapper = new QueryWrapper<>();  //SELECT id,user\_name,password,name,age,email FROM tb\_user WHERE name LIKE ?  //Parameters: %曹%(String)  wrapper.likeRight("name", "曹");  wrapper.select("id","name");  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.out.println(user);  }  } |

Lambda方式构造条件：



### 6.3.逻辑查询

1）or

拼接OR

主动调用 or 表示紧接着下一个方法不是用 and 连接!(不调用 or 则默认为使用and 连接)

2）and

AND 嵌套

例: and(i -> i.eq("name", "李白").ne("status", "活着")) ---> and (name = '李白' and status <> '活着')

测试用例：

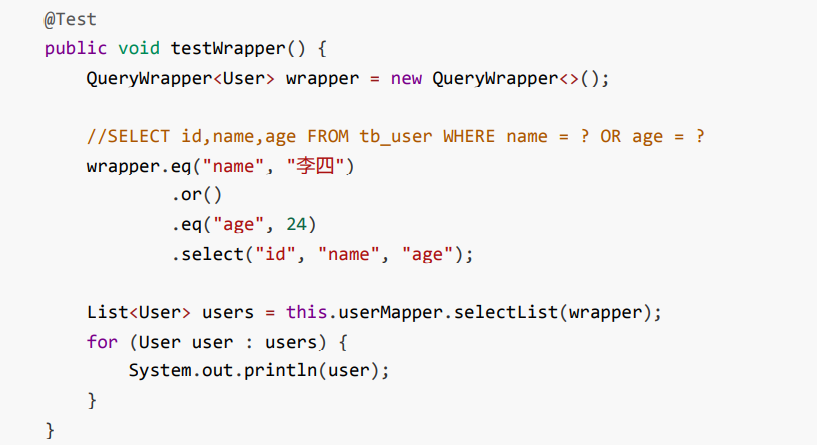
|  |
| --- |
| @Test  public void testWrapper() {  QueryWrapper wrapper = new QueryWrapper<>();  //SELECT id,user\_name,password,name,age,email FROM tb\_user WHERE name = ? OR age = ? wrapper.eq("name","李四").or().eq("age", 24);  //变为and方式  wrapper.eq("name","李四").eq("age", 24)  List users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.out.println(user);  } } |

Lambda方式构造条件：

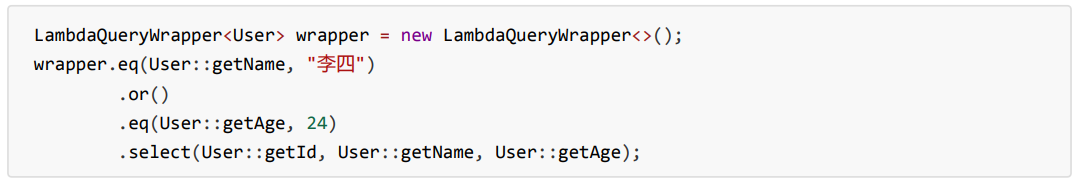


### 6.4.select

在MP查询中，默认查询所有的字段，如果有需要也可以通过select方法进行指定字段。



Lambda方式构造条件：



### 6.5.排序

1）orderByAsc

**升序排序**

参数：变长数组，设置多个字段名

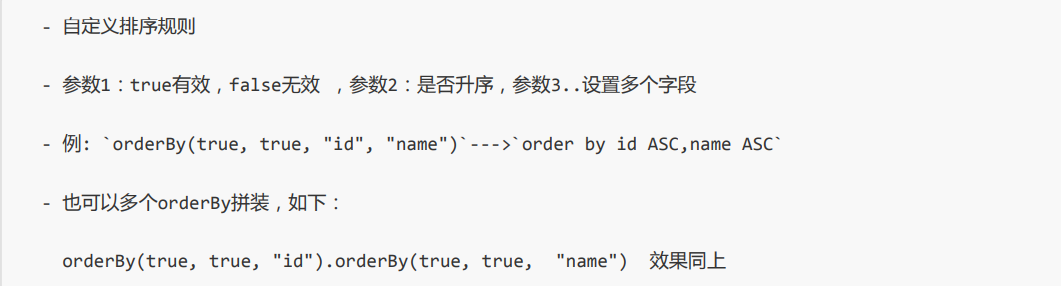
例: orderByAsc("id", "name") ---> order by id ASC,name ASC

1. orderByDesc 降序排序

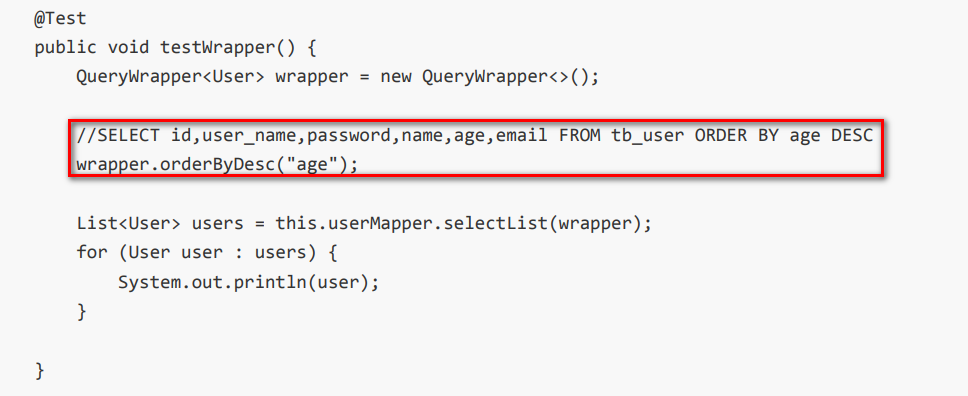
参数：变长数组，设置多个字段名

例: orderByDesc("id", "name") ---> order by id DESC,name DESC

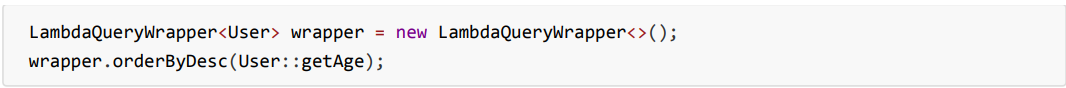
3）orderBy orderBy(boolean condition, boolean isAsc, R... columns)



例子：



Lambda方式构造条件：



## 7.代码生成器

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Mapper接口、 Entity实体类及Mapper XML文件、 Service 、Controller 等各个模块的代码，极大的提升了开发效率。

1）在pom文件中引入依赖

|  |
| --- |
| <!-- MyBatis-plus相关依赖 -->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus</artifactId>  <version>3.1.2</version>  </dependency> |

2）获取案例

|  |
| --- |
| package com.yimupay.generator;  import com.baomidou.mybatisplus.annotation.DbType;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;  import com.baomidou.mybatisplus.core.toolkit.StringPool;  import com.baomidou.mybatisplus.core.toolkit.StringUtils;  import com.baomidou.mybatisplus.generator.AutoGenerator;  import com.baomidou.mybatisplus.generator.InjectionConfig;  import com.baomidou.mybatisplus.generator.config.\*;  import com.baomidou.mybatisplus.generator.config.po.TableInfo;  import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;  import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;  import lombok.extern.slf4j.Slf4j;  import java.util.ArrayList;  import java.util.List;  import java.util.Scanner;  /\*\*  \* <p>  \* MyBatis Plus Generator 配置执行类示例  \* </p>  \*  \* @author  \* @since  \*/  @Slf4j  public class MyBatisPlusGenerator {    //是否生成DTO  private static final Boolean IS\_DTO = true;    /\*\*  \* <p>  \* 读取控制台内容,输入模块名  \* </p>  \*/  public static String scanner(String tip) {  Scanner scanner = new Scanner(System.in);  StringBuilder help = new StringBuilder();  help.append("请输入" + tip + "：");  System.out.println(help.toString());  if (scanner.hasNext()) {  String ipt = scanner.next();  if (StringUtils.isNotEmpty(ipt)) {  return ipt;  }  }  throw new MybatisPlusException("请输入正确的" + tip + "！");  }  public static void main(String[] args) {  // 代码生成器  AutoGenerator autoGenerator = new AutoGenerator();  // 指定模板引擎为freemaker, 默认为Velocity  autoGenerator.setTemplateEngine(new FreemarkerTemplateEngine());    // 全局配置  GlobalConfig globalConfig = new GlobalConfig();  globalConfig.setFileOverride(true);//是否覆盖文件  String projectPath = System.getProperty("user.dir");  //当前项目名  String projectName = "/generator";  globalConfig.setOutputDir(projectPath + projectName+"/src/main/java");// 输出路径  log.info("文件生成路径：" + projectPath + projectName+"/src/main/java");  globalConfig.setAuthor("author");//作者名称  globalConfig.setOpen(false);//生成后是否自动打开文件  globalConfig.setIdType(IdType.ID\_WORKER);  globalConfig.setServiceName("%sService");//生成的service接口名称  globalConfig.setServiceImplName("%sServiceImpl");//生成的service接口实现名称  globalConfig.setBaseResultMap(true);//mapper.xml中生成基础resultMap  globalConfig.setBaseColumnList(true);//mapper.xml中生成基础columnList  autoGenerator.setGlobalConfig(globalConfig);  // 数据源配置 需配置  DataSourceConfig dataSourceConfig = new DataSourceConfig();  // 商户服务  // dataSourceConfig  // .setUrl("jdbc:mysql://218.29.75.103:6306/shanjupay\_merchant\_service?serverTimezone=Asia/Shanghai");  // 交易服务  dataSourceConfig  .setUrl("jdbc:mysql://localhost:3306/yimupay\_merchant\_service?useSSL=false&serverTimezone=Asia/Shanghai");  // dataSourceConfig.setSchemaName("public");  dataSourceConfig.setDbType(DbType.MYSQL);//数据库类型  dataSourceConfig.setDriverName("com.mysql.cj.jdbc.Driver");  dataSourceConfig.setUsername("root");  dataSourceConfig.setPassword("mnj852123");  autoGenerator.setDataSource(dataSourceConfig);  // 生成包配置  PackageConfig packageConfig = new PackageConfig();  packageConfig.setParent("com.me.yimu.pay");  //如果需要手动输入模块名  packageConfig.setModuleName(scanner("模块名"));//模块名，画面输入  packageConfig.setServiceImpl("service");  packageConfig.setXml("mapper");  autoGenerator.setPackageInfo(packageConfig);  // 生成DTO文件配置  if (IS\_DTO) {  globalConfig.setSwagger2(true);//是否使用swagger2  globalConfig.setEntityName("%sDTO");  packageConfig.setEntity("dto");  }    // 自定义配置  InjectionConfig injectionConfig = new InjectionConfig() {  @Override  public void initMap() {  // to do nothing  }  };  // 如果模板引擎是 freemarker  String templatePath = "/templates/mapper.xml.ftl";  // 自定义输出配置  List<FileOutConfig> focList = new ArrayList<>();  // 自定义配置会被优先输出  focList.add(new FileOutConfig(templatePath) {  @Override  public String outputFile(TableInfo tableInfo) {  // 自定义输出文件名  return projectPath + projectName+"/src/main/resources/mapper/" + packageConfig.getModuleName()  + "/" + tableInfo.getEntityName() + "Mapper" + StringPool.DOT\_XML;  }  });  injectionConfig.setFileOutConfigList(focList);  autoGenerator.setCfg(injectionConfig);  // 配置模板  TemplateConfig templateConfig = new TemplateConfig();  // 配置自定义输出模板  //指定自定义模板路径，注意不要带上.ftl/.vm, 会根据使用的模板引擎自动识别  // templateConfig.setEntity("templates/entity-test.java");  // templateConfig.setService("templates/service.java");  // templateConfig.setController("templates/controller.java");  templateConfig.setXml(null);  autoGenerator.setTemplate(templateConfig);  // 策略配置  StrategyConfig strategyConfig = new StrategyConfig();  strategyConfig.setNaming(NamingStrategy.underline\_to\_camel);//表名映射到实体策略，带下划线的转成驼峰  strategyConfig.setColumnNaming(NamingStrategy.underline\_to\_camel);//列名映射到类型属性策略，带下划线的转成驼峰  // strategyConfig.setSuperEntityClass("com.baomidou.ant.common.BaseEntity");  strategyConfig.setEntityLombokModel(true);//实体类使用lombok  // strategyConfig.setRestControllerStyle(true);  // strategyConfig.setSuperControllerClass("com.baomidou.ant.common.BaseController");  // 如果 setInclude() //设置表名不加参数, 会自动查找所有表  // 如需要制定单个表, 需填写参数如: strategyConfig.setInclude("user\_info);  strategyConfig.setInclude();  // strategyConfig.setSuperEntityColumns("id");  // strategyConfig.setControllerMappingHyphenStyle(true);  //自动将数据库中表名为 user\_info 格式的转为 UserInfo 命名  strategyConfig.setTablePrefix(packageConfig.getModuleName() + "\_");//表名映射到实体名称去掉前缀  strategyConfig.setEntityBooleanColumnRemoveIsPrefix(true);// Boolean类型字段是否移除is前缀处理  autoGenerator.setStrategy(strategyConfig);    System.out.println("===================== MyBatis Plus Generator ==================");  autoGenerator.execute();  System.out.println("================= MyBatis Plus Generator Execute Complete ==================");  }  } |

3）测试

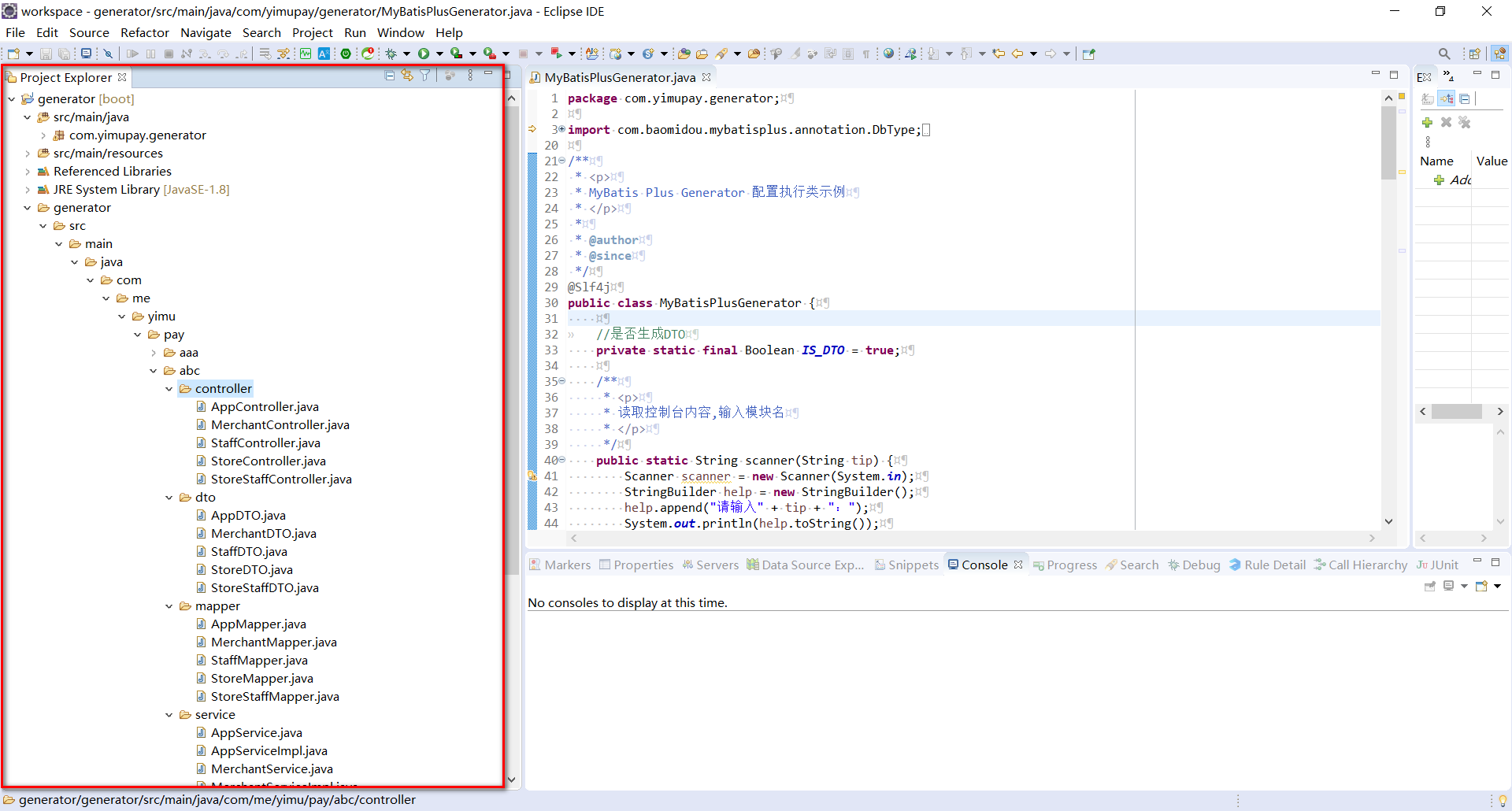
运行MyBatisPlusGenerator的main方法。

输入模块名：

**注意：模块名匹配表名前缀会自动去掉，否则生成的模型类保留前缀。**

|  |
| --- |
| 22:31:07.971 [main] INFO com.yimupay.generator.MyBatisPlusGenerator - 文件生成路径：D:\projects\generator\generator/generator/src/main/java  **请输入模块名：**  **abc**  ===================== MyBatis Plus Generator ==================  22:31:17.006 [main] DEBUG com.baomidou.mybatisplus.generator.AutoGenerator - ==========================准备生成文件...==========================  22:31:17.298 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto]  22:31:17.298 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller]  22:31:17.299 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service]  22:31:17.299 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper]  22:31:17.375 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl; 文件:D:\projects\generator\generator/generator/src/main/resources/mapper/abc/AppDTOMapper.xml  22:31:17.453 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/entity.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto\AppDTO.java  22:31:17.455 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper\AppMapper.java  22:31:17.457 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/service.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\AppService.java  22:31:17.461 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/serviceImpl.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\AppServiceImpl.java  22:31:17.464 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/controller.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller\AppController.java  22:31:17.466 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl; 文件:D:\projects\generator\generator/generator/src/main/resources/mapper/abc/MerchantDTOMapper.xml  22:31:17.470 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/entity.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto\MerchantDTO.java  22:31:17.471 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper\MerchantMapper.java  22:31:17.471 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/service.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\MerchantService.java  22:31:17.472 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/serviceImpl.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\MerchantServiceImpl.java  22:31:17.472 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/controller.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller\MerchantController.java  22:31:17.473 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl; 文件:D:\projects\generator\generator/generator/src/main/resources/mapper/abc/StaffDTOMapper.xml  22:31:17.474 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/entity.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto\StaffDTO.java  22:31:17.475 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper\StaffMapper.java  22:31:17.476 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/service.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StaffService.java  22:31:17.476 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/serviceImpl.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StaffServiceImpl.java  22:31:17.477 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/controller.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller\StaffController.java  22:31:17.478 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl; 文件:D:\projects\generator\generator/generator/src/main/resources/mapper/abc/StoreDTOMapper.xml  22:31:17.480 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/entity.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto\StoreDTO.java  22:31:17.481 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper\StoreMapper.java  22:31:17.481 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/service.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StoreService.java  22:31:17.482 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/serviceImpl.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StoreServiceImpl.java  22:31:17.484 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/controller.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller\StoreController.java  22:31:17.485 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl; 文件:D:\projects\generator\generator/generator/src/main/resources/mapper/abc/StoreStaffDTOMapper.xml  22:31:17.486 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/entity.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\dto\StoreStaffDTO.java  22:31:17.486 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\mapper\StoreStaffMapper.java  22:31:17.487 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/service.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StoreStaffService.java  22:31:17.488 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/serviceImpl.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\service\StoreStaffServiceImpl.java  22:31:17.489 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/controller.java.ftl; 文件:D:\projects\generator\generator/generator/src/main/java\com\me\yimu\pay\abc\controller\StoreStaffController.java  22:31:17.489 [main] DEBUG com.baomidou.mybatisplus.generator.AutoGenerator - ==========================文件生成完成！！！==========================  ================= MyBatis Plus Generator Execute Complete ================== |

生成的文件包括entity、controller、service、mapper，如下：



4）自定义模板

上边的自动生成代码是通过freemarker引擎生成代码，可以通过自定义freemarker模板对生成代码进行个性化定义。

在resources下创建templates目录，目录下放入要个性化定义模板即可。

具体模板参考资料文件夹下的templates目录 。

## 8.参照

https://baijiahao.baidu.com/s?id=1628341991548391304&wfr=spider&for=pc