Mybatis-Plus

课程目标：

- 了解Mybatis-Plus
- 整合Mybatis-Plus
- 通用CRUD
- Mybatis-Plus的配置
- 条件构造器
- Mybatis-Plus 的Service封装
- 代码生成器

1 Mybatis-Plus介绍

1.1 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

> 愿景

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

1.2 支持的数据库

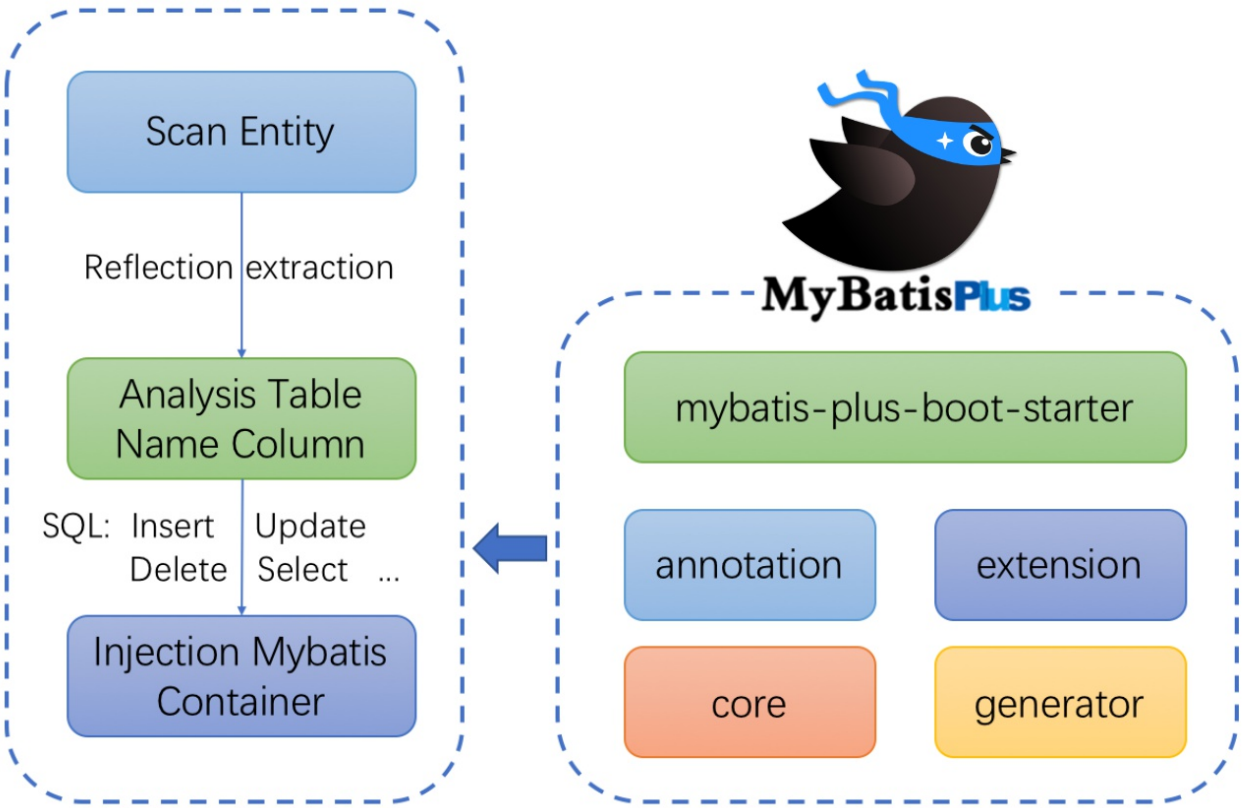
MyBatisPlus支持如下数据库：

- 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

1.3 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

1.4 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)。

2 快速入门

2.1 准备环境

- JDK 8+
- Maven 3.3.9
- IDEA 2018.2

- MySQL5.7

2.2 创建数据库以及表

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');
```

2.3 工程搭建

2.3.1 创建工程

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

```
    <groupId>cn.itcast.mp</groupId>
    <artifactId>itcast-mp-springboot</artifactId>
    <version>1.0-SNAPSHOT</version>
```

2.3.2 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>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.3.RELEASE</version>
    </parent>

    <groupId>cn.itcast.mp</groupId>
    <artifactId>itcast-mp-springboot</artifactId>
    <version>1.0-SNAPSHOT</version>

    <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>
```

2.3.3 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=root

# Logger Config
logging.level.root: debug
```

2.3.4 编写pojo

```
package cn.itcast.mp.pojo;

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

import java.time.LocalDateTime;

@Data
@NoArgsConstructor
@AllArgsConstructor
@TableName("tb_user")
public class User {
    @TableId("ID")
    private Long id;
    @TableField("USER_NAME")
    private String userName; //驼峰命名,则无需注解
    @TableField("PASSWORD")
    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

2.3.5 编写mapper

```
package cn.itcast.mp.mapper;

import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;

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

2.3.6 编写启动类

```
package cn.itcast.mp;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;

@MapperScan("cn.itcast.mp.mapper") //设置mapper接口的扫描包
@SpringBootApplication
public class MyApplication {

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

}
```

2.3.7 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 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.util.List;
```

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelect() {
        List<User> userList = userMapper.selectList(null);
        for (User user : userList) {
            System.out.println(user);
        }
    }

}
```

测试：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters:
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <==      Total: 5
[main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional SqlSession
[org.apache.ibatis.session.defaults.DefaultSqlSession@14faa38c]
User(id=1, userName=zhangsan, password=123456, name=张三, age=18, email=test1@itcast.cn,
birthday=2019-09-26T11:42:01)
User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
birthday=2019-10-01T11:42:08)
User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn,
birthday=2019-10-02T11:42:14)
User(id=4, userName=zhaoliu, password=123456, name=赵六, age=21, email=test4@itcast.cn,
birthday=2019-10-05T11:42:18)
User(id=5, userName=sunqi, password=123456, name=孙七, age=24, email=test5@itcast.cn,
birthday=2019-10-14T11:42:23)
```

3 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。

3.1 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>

    <!--<plugins>-->
        <!--<plugin
interceptor="com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor"></plugin>-->
    <!--</plugins>-->

</configuration>
```

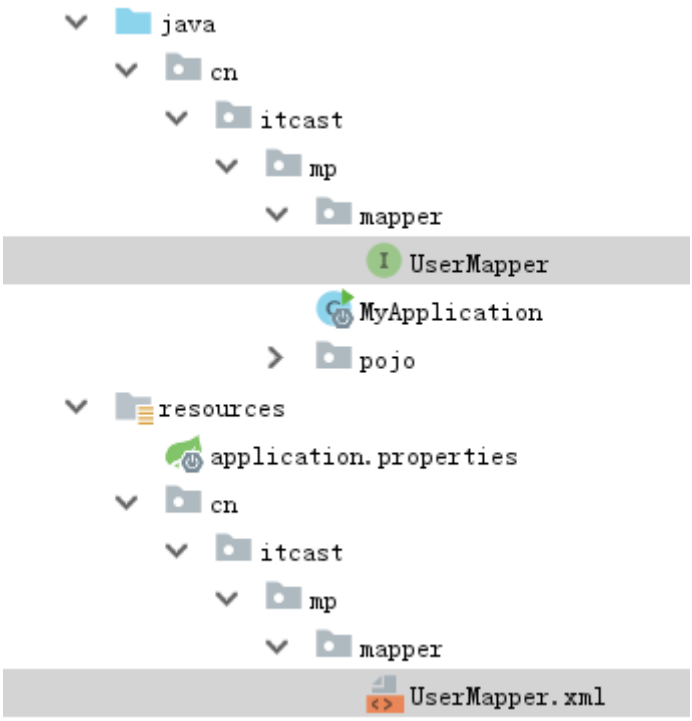
2 在application.properties下配置configLocations，如下：

```
mybatis-plus.config-location = classpath:mybatis-config.xml
```

3.2 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
<?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="cn.itcast.mp.pojo.User" parameterType="java.lang.Long">
      select * from tb_user where id = #{id}
    </select>

</mapper>
```

```java
//在UserMapper接口类中新增findById方法
package cn.itcast.mp.mapper;

import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;

public interface UserMapper extends BaseMapper<User> {

    User findById(Long id);
}
```

测试用例：

```java
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelectPage() {
        User user = this.userMapper.findById(2L);
        System.out.println(user);
    }
}
```

运行结果：

```
ansaction]-[DEBUG] JDBC Connection [HikariProxyConnection@9063
G] ==>  Preparing: select * from tb_user where id = ?
G] ==> Parameters: 2(Long)
G] <==      Total: 1
osing non transactional SqlSession [org.apache.ibatis.session.
ge=20, email=test2@itcast.cn, address=null)
```

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）**

3.3 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
```

3.4 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
```

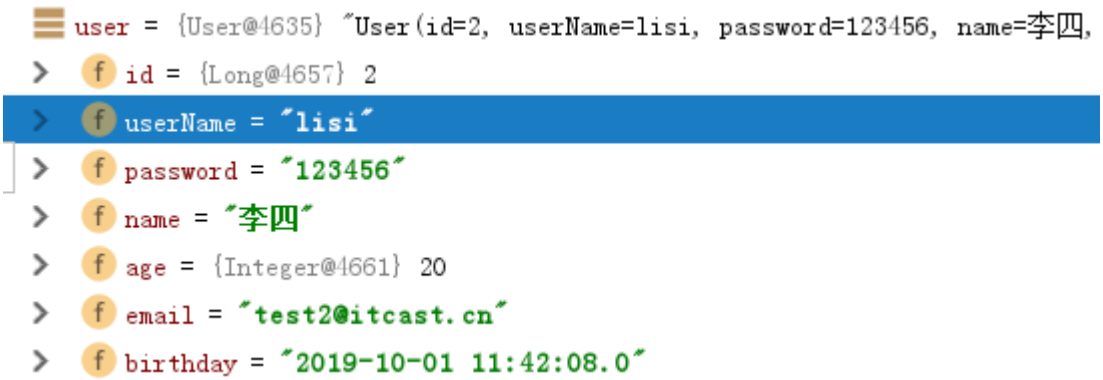
1 屏蔽@TableField

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@TableName("tb_user")
public class User {

    @TableId("ID")
```

```
    private Long id;
//    @TableField("USER_NAME")
    private String userName; //驼峰命名,则无需注解
//    @TableField("PASSWORD")
    private String password;
//    @TableField("NAME")
    private String name;
//    @TableField("AGE")
    private Integer age;
//    @TableField("EMAIL")
    private String email;
//    @TableField("BIRTHDAY")
    private LocalDateTime birthday;
}
```

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：

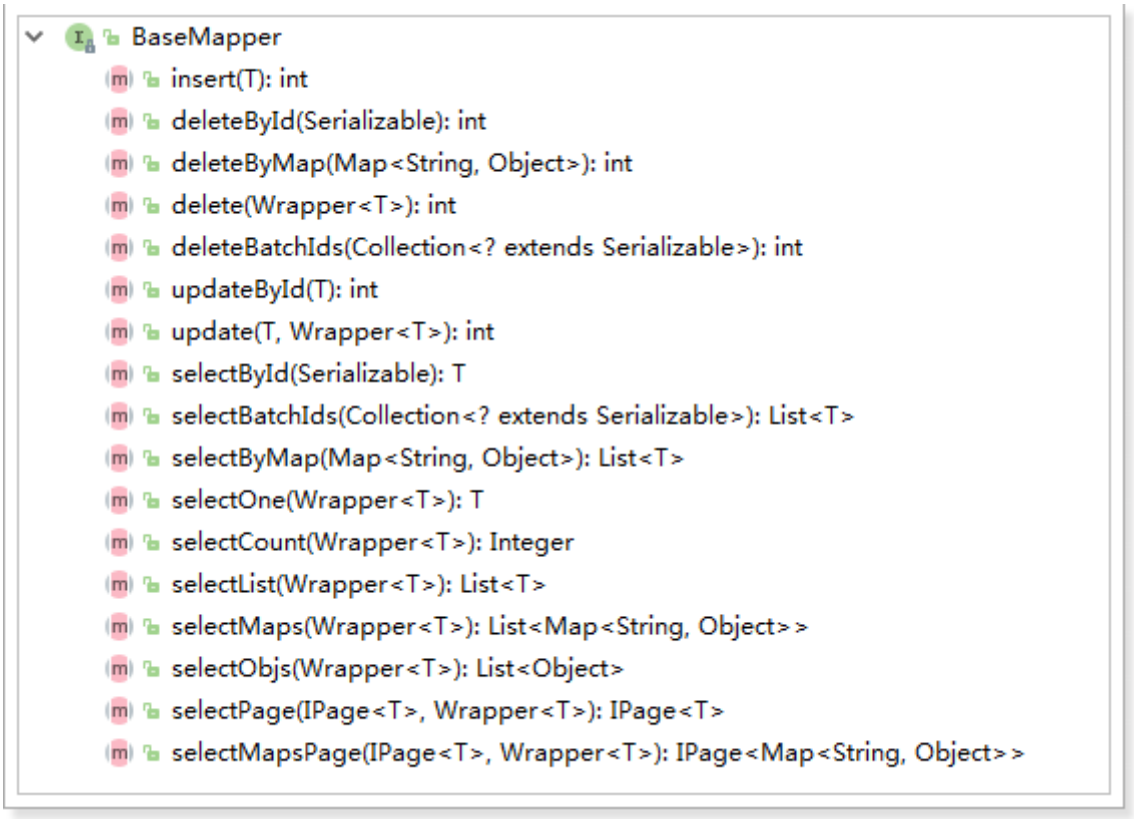
只使用一种配置方法。

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

```
<settings>
    <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
```

4 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的各各方法：



4.1 插入操作

4.1.1 方法定义

```
/**
 * 插入一条记录
 *
 * @param entity 实体对象
 */
int insert(T entity);
```

4.1.2 测试用例

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {
```

```
    @Autowired
    private UserMapper userMapper;

   @Test
    public void testInsert(){
        User user = new User();
        user.setAge(20);
        user.setEmail("test@itcast.cn");
        user.setName("曹操");
        user.setUserName("caocao");
        user.setPassword("123456");
        DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        LocalDateTime localDateTime = LocalDateTime.parse("1990-01-01 00:00:00", df);
        user.setBirthday(localDateTime);
        int result = this.userMapper.insert(user); //返回的result是受影响的行数，并不是自增后的id
        System.out.println("result = " + result);

        System.out.println(user.getId()); //自增后的id会回填到对象中
    }
}
```

4.1.3 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：

```
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),

    /**
     * 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。

1) 设置mysql数据库主键为自增

2) 修改User对象：

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

3) 程序中不用设置主键

2 输入主键：

手动设置主键值。

1) mysql数据库主键为自增或不是自增都可以

2) 修改User对象：

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

3) 程序中需要设置主键

3 UUID：

生成全局唯一ID。

1）mysql数据库主键为字符串类型，不是自增类型。

2）修改User对象。

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

3）程序中不用设置主键

4 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。

1）mysql数据库主键为字符串类型，不是自增类型。

2）修改User对象。

```
@TableId(value = "ID",type = IdType.ID_WORKER_STR)
private String id;
```

3）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。

1）mysql数据库主键为数值类型，不是自增类型。

2）修改User对象。

```
@TableId(value = "ID",type = IdType.ID_WORKER)
private Long id;
```

3）程序中不用设置主键

4.2 更新操作

4.2.1 根据id更新

方法定义：

```
    /**
     * 根据 ID 修改
     *
     * @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：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testUpdateById() {
        User user = new User();
        user.setId(6L); //主键
        user.setAge(21); //更新的字段

        //根据id更新，更新不为null的字段
        this.userMapper.updateById(user);
    }
}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==>  Preparing: UPDATE tb_user SET
age=? WHERE id=?
[main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Parameters: 21(Integer), 6(Long)
[main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] <==    Updates: 1
```

4.2.2 根据条件更新

方法定义：

```
    /**
     * 根据 whereEntity 条件，更新记录
     *
     * @param entity        实体对象 (set 条件值,可以为 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。

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.Wrapper;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;
import net.minidev.json.writer.UpdaterMapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testUpdate() {
        User user = new User();
        user.setAge(22); //更新的字段
```

```
        //更新的条件
        QueryWrapper<User> wrapper = new QueryWrapper<>();
        wrapper.eq("name", "曹操");
        //可以设置多个条件...

        //执行更新操作
        int result = this.userMapper.update(user, wrapper);
        System.out.println("result = " + result);
    }

}
```

上边根据id更新 根据条件更新的方法只能将对象中不为NULL的属性更新到表中，下边通过UpdateWrapper进行更新，将birthday字段更新为NULL.

```
    @Test
    public void testUpdate2() {
        //更新的条件以及字段
        UpdateWrapper<User> wrapper = new UpdateWrapper<>();
        wrapper.eq("id", 6).set("age", 23).set("birthday",null);

        //执行更新操作
        int result = this.userMapper.update(null, wrapper);
        System.out.println("result = " + result);
    }
```

4.3 删除操作

4.3.1 deleteById

方法定义：

```
/**
 * 根据 ID 删除
 *
 * @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：

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

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

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==>  Preparing: DELETE FROM tb_user
WHERE id=?
[main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Parameters: 6(Long)
[main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] <==    Updates: 1
```

4.3.2 delete

方法定义：

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

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

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.QueryWrapper;
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.util.HashMap;
import java.util.Map;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testDeleteByMap() {
        User user = new User();
        user.setAge(20);
        user.setName("张三");

        //将实体对象进行包装，包装为操作条件
        QueryWrapper<User> wrapper = new QueryWrapper<>(user);

        int result = this.userMapper.delete(wrapper);
        System.out.println("result = " + result);
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] ==>  Preparing: DELETE FROM tb_user WHERE
name=? AND age=?
[main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] ==> Parameters: 张三(String), 20(Integer)
[main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] <==    Updates: 0
```

注意：

定义QueryWrapper可以不包装模型对象，手动设置条件，如下：

```
QueryWrapper<User> wrapper = new QueryWrapper<>();
wrapper.eq("age",20);
wrapper.eq("name","张三");
```

4.3.3 deleteBatchIds

方法定义：

```
    /**
     * 删除（根据ID 批量删除）
     *
     * @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：

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
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.util.Arrays;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testDeleteByMap() {
        //根据id集合批量删除
        int result = this.userMapper.deleteBatchIds(Arrays.asList(1L,10L,20L));
        System.out.println("result = " + result);
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] ==>  Preparing: DELETE FROM
tb_user WHERE id IN ( ? , ? , ? )
[main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] ==> Parameters: 1(Long),
10(Long), 20(Long)
[main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] <==    Updates: 1
```

4.4 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。

4.4.1 selectById

方法定义：

```
    /**
     * 根据 ID 查询
     *
     * @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：

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelectById() {

        //根据id查询数据
```

```
        User user = this.userMapper.selectById(2L);
        System.out.println("result = " + user);
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE id=?
[main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Parameters: 2(Long)
[main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] <==      Total: 1

result = User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
address=null)
```

4.4.2 selectBatchIds

方法定义：

```
/**
 * 查询（根据ID 批量查询）
 *
 * @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：

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
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.util.Arrays;
import java.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {
```

```
    @Autowired
    private UserMapper userMapper;

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

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE id IN ( ? , ? , ? )
[main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] ==> Parameters: 2(Long), 3(Long),
10(Long)
[main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] <==      Total: 2

User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
address=null)
User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn,
address=null)
```

4.4.3 selectOne

方法定义：

```
/**
 * 根据 entity 条件，查询一条记录
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 */
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：

```
package cn.itcast.mp;
```

```
import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelectOne() {
        QueryWrapper<User> wrapper = new QueryWrapper<User>();
        wrapper.eq("name", "李四");

        //根据条件查询一条数据，如果结果超过一条会报错
        User user = this.userMapper.selectOne(wrapper);
        System.out.println(user);
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE name = ?
[main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] ==> Parameters: 李四(String)
[main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] <==      Total: 1

User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
address=null)
```

4.4.4 selectCount

方法定义：

```
/**
 * 根据 Wrapper 条件，查询总记录数
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 */
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

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.QueryWrapper;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelectCount() {
        QueryWrapper<User> wrapper = new QueryWrapper<User>();
        wrapper.gt("age", 23); //年龄大于23岁

        //根据条件查询数据条数
        Integer count = this.userMapper.selectCount(wrapper);
        System.out.println("count = " + count);
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] ==>  Preparing: SELECT COUNT( 1 )
FROM tb_user WHERE age > ?
[main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] ==> Parameters: 23(Integer)
[main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] <==      Total: 1

count = 2
```

4.4.5 selectList

方法定义：

```
/**
 * 根据 entity 条件，查询全部记录
 *
 * @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：

```
package cn.itcast.mp;
```

```
import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSelectList() {
        QueryWrapper<User> wrapper = new QueryWrapper<User>();
        wrapper.gt("age", 23); //年龄大于23岁

        //根据条件查询数据
        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println("user = " + user);
        }
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE age > ?
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters: 23(Integer)
[main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <==      Total: 2

user = User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn,
address=null)
user = User(id=5, userName=sunqi, password=123456, name=孙七, age=24, email=test5@itcast.cn,
address=null)
```

4.4.6 selectPage

方法定义：

```
/**
 * 根据 entity 条件，查询全部记录（并翻页）
 *
 * @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;

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;

@Configuration
@MapperScan("cn.itcast.mp.mapper") //设置mapper接口的扫描包
public class MybatisPlusConfig {

    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }
}
```

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.QueryWrapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
```

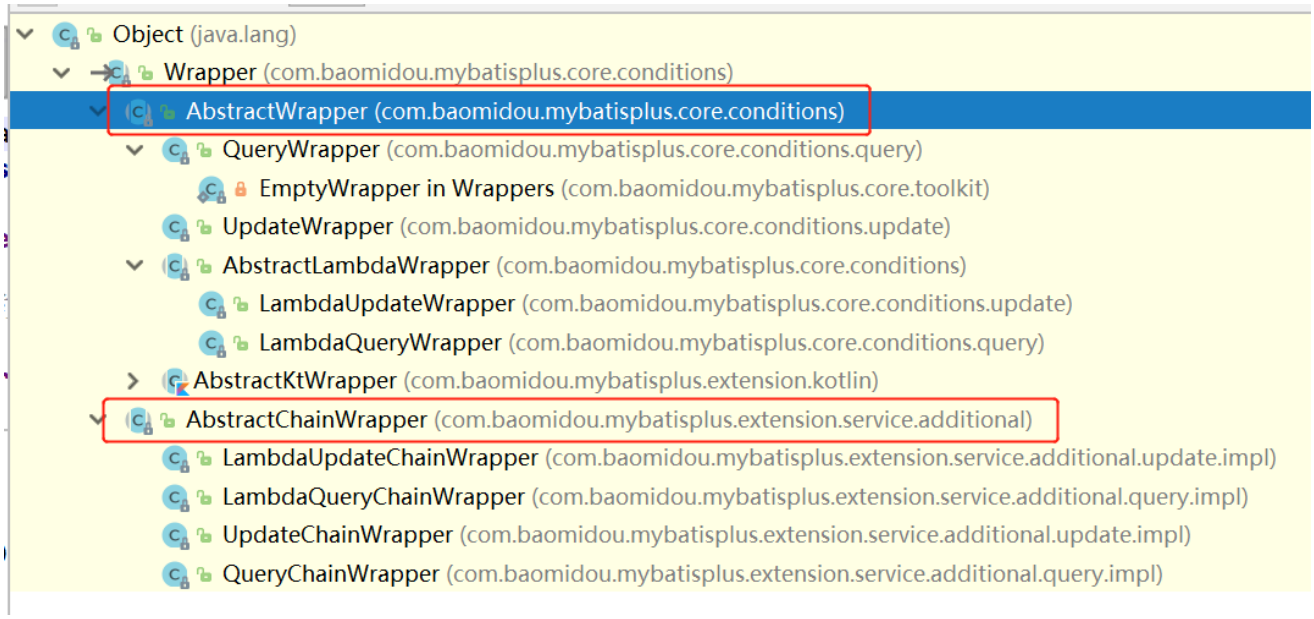
```
    private UserMapper userMapper;

    @Test
    public void testSelectPage() {
        QueryWrapper<User> wrapper = new QueryWrapper<User>();
        wrapper.gt("age", 20); //年龄大于20岁

        //参数1：当前页码，小于1的按1算
        //参数2：每页记录数
        Page<User> page = new Page<>(1,1);

        //根据条件查询数据
        IPage<User> iPage = this.userMapper.selectPage(page, wrapper);
        System.out.println("数据总条数：" + iPage.getTotal());
        System.out.println("总页数：" + iPage.getPages());

         //取出分页记录
        List<User> users = iPage.getRecords();
        for (User user : users) {
            System.out.println("user = " + user);
        }
    }

}
```

结果：

```
[main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==>  Preparing: SELECT COUNT(1) FROM
tb_user WHERE age > ?
[main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Parameters: 20(Integer)
[main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==>  Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE age > ? LIMIT ?,?
[main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Parameters: 20(Integer), 0(Long),
1(Long)
[main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] <==      Total: 1
[main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional SqlSession
[org.apache.ibatis.session.defaults.DefaultSqlSession@6ecd665]
数据总条数：3
总页数：3
user = User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn,
birthday=2019-10-02T11:42:14, address=null)
```

5 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。

5.1 基本比较操作

- eq
  - 等于 =
- ne
  - 不等于 <>
- gt
  - 大于 >
- ge
  - 大于等于 >=
- lt
  - 小于 <
- le
  - 小于等于 <=
- between
  - BETWEEN 值1 AND 值2
- notBetween
  - NOT BETWEEN 值1 AND 值2
- in
  - 字段 IN (value.get(0), value.get(1), ...)
- notIn
  - 字段 NOT IN (v0, v1, ...)

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.QueryWrapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testEq() {
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        //SELECT id,user_name,password,name,age,email FROM tb_user WHERE password = ? AND age >=
? AND name IN (?,?,?)
        wrapper.eq("password", "123456")
                .ge("age", 20)
                .in("name", "李四", "王五", "赵六");

        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);
        }

    }
```

Lambda方式构造条件：

```
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();

        //SELECT id,user_name,password,name,age,email FROM tb_user WHERE password = ? AND age >=
? AND name IN (?,?,?)
        wrapper.eq(User::getPassword, "123456")
                .ge(User::getAge, 20)
                .in(User::getName, "李四", "王五", "赵六");

        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);

        }
```

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：

```
eq(boolean condition, R column, Object val)
in(boolean condition, R column, Object... values)
...
```

一个例子：

```
//比如根据name来判断，如果name不为空则拼接条件
String name = null;
wrapper.eq(User::getPassword, "123456")
        .ge(User::getAge, 20)
        .in(name!=null,User::getName, "李四", "王五", "赵六");
```

5.2 模糊查询

- like
  - LIKE '%值%'
  - 例: `like("name", "王")` ---> `name like '%王%'`
- notLike
  - NOT LIKE '%值%'
  - 例: `notLike("name", "王")` ---> `name not like '%王%'`
- likeLeft
  - LIKE '%值'
  - 例: `likeLeft("name", "王")` ---> `name like '%王'`
- likeRight
  - LIKE '值%'
  - 例: `likeRight("name", "王")` ---> `name like '王%'`

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.QueryWrapper;
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.util.List;
```

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testWrapper() {
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        //SELECT id,user_name,password,name,age,email FROM tb_user WHERE name LIKE ?
        //Parameters: %五%(String)
        wrapper.like("name", "五");

        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);
        }

    }

}
```

Lambda方式构造条件：

```
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
wrapper.like(User::getName, "五");
```

5.3 逻辑查询

- 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)
- 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：

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testWrapper() {
        QueryWrapper<User> 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<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);
        }

    }

}
```

Lambda方式构造条件：

```
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
wrapper.eq(User::getName,"李四").or().eq(User::getAge, 24);
```

5.4 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。

```
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
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.util.List;
```

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testWrapper() {
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        //SELECT id,name,age FROM tb_user WHERE name = ? OR age = ?
        wrapper.eq("name", "李四")
                .or()
                .eq("age", 24)
                .select("id", "name", "age");

        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);
        }
    }
}
```

Lambda方式构造条件：

```
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
wrapper.eq(User::getName, "李四")
        .or()
        .eq(User::getAge, 24)
        .select(User::getId, User::getName, User::getAge);
```

5.5 排序

- 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`
- 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`
- orderBy

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

- 自定义排序规则

- 参数1：true有效，false无效 ，参数2：是否升序，参数3..设置多个字段

- 例: `orderBy(true, true, "id", "name")`--->`order by id ASC,name ASC`

- 也可以多个orderBy拼装，如下：

  orderBy(true, true, "id").orderBy(true, true, "name") 效果同上

例子：

```java
package cn.itcast.mp;

import cn.itcast.mp.mapper.UserMapper;
import cn.itcast.mp.pojo.User;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
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.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testWrapper() {
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        //SELECT id,user_name,password,name,age,email FROM tb_user ORDER BY age DESC
        wrapper.orderByDesc("age");

        List<User> users = this.userMapper.selectList(wrapper);
        for (User user : users) {
            System.out.println(user);
        }

    }

}
```

Lambda方式构造条件：

```
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
wrapper.orderByDesc(User::getAge);
```

6 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

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.1.0</version>
</dependency>
```

2）获取官方案例

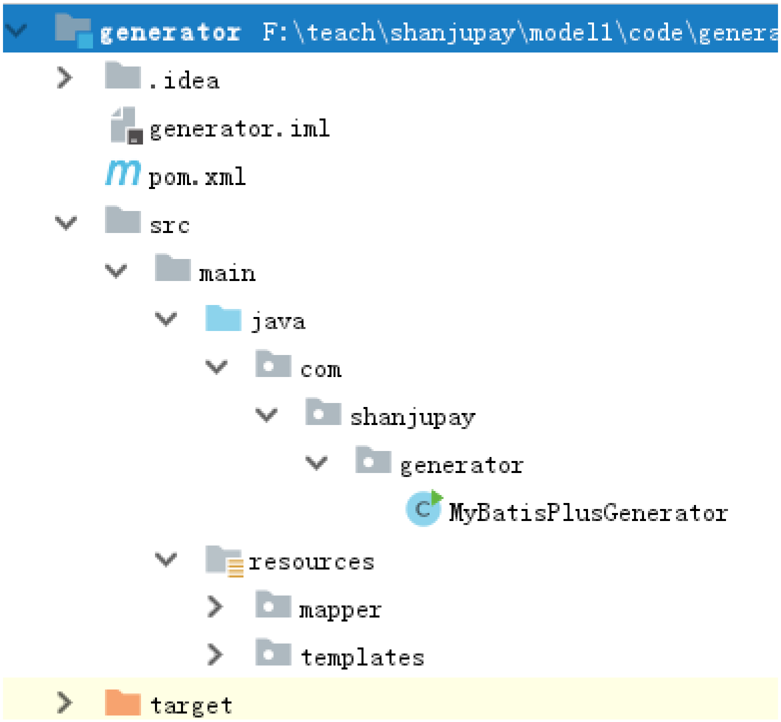
```
package cn.itcast.mp.generator;

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

/**
 * <p>
 * 代码生成器演示例子
 * </p>
 */
public class MpGenerator {

    // TODO 修改服务名以及数据表名
    private static final String SERVICE_NAME = "content";
    private static final String[] TABLE_NAMES = new String[]{
            "tb_user",

            "t_user"
```

```
    };

    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();
        // 选择 freemarker 引擎，默认 Velocity
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        gc.setFileOverride(true);//是否覆盖文件
        gc.setOutputDir(System.getProperty("user.dir") + "/src/main/java");//输出路径
        gc.setAuthor("itcast");//作者名称
        gc.setOpen(false);//生成后是否自动打开文件
        gc.setSwagger2(false);//是否使用swagger2
        gc.setServiceName("%sService");//生成的service接口名称
        gc.setServiceImplName("%sServiceImpl");
        gc.setBaseResultMap(true);//mapper.xml中生成基础resultMap
        gc.setBaseColumnList(true);//mapper.xml中生成基础columnList

        mpg.setGlobalConfig(gc);

        // 数据库配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setDbType(DbType.MYSQL);//数据类型
        dsc.setUrl("jdbc:mysql://127.0.0.1:3306/account?useUnicode=true&characterEncoding=UTF-
8&serverTimezone=UTC&useSSL=false");
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("mysql");
        mpg.setDataSource(dsc);

        // 包配置
        PackageConfig pc = new PackageConfig();
        pc.setModuleName(SERVICE_NAME);
        pc.setParent("cn.itcast");
        pc.setServiceImpl("service");
        pc.setXml("mapper");
        mpg.setPackageInfo(pc);

        // 设置模板
        TemplateConfig tc = new TemplateConfig();
        mpg.setTemplate(tc);

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);//表名映射到实体策略，带下划线的转成
驼峰
        strategy.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
        strategy.setEntityLombokModel(true);//实体类使用lombok
//      strategy.setRestControllerStyle(true);//controller使用rest接口模式
        strategy.setInclude(TABLE_NAMES);//设置表名

//      strategy.setTablePrefix("tb_");//表名映射到实体名称去掉前缀
```

```java
        strategy.setEntityBooleanColumnRemoveIsPrefix(true);// Boolean类型字段是否移除is前缀处理

        mpg.setStrategy(strategy);

        mpg.execute();
    }

}
```

3）测试

这里使用提供的生成器工程，解压generator.zip，并导入IDEA。



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。

```
请输入模块名：
tb
==================== MyBatis Plus Generator =================
12:30:52.413 [main] DEBUG com.baomidou.mybatisplus.generator.AutoGenerator - =========================准备生成文件...==========================
Fri Nov 29 12:30:52 CST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.
12:30:52.771 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [F:\teach\shanjupay\model1\code\generator/
12:30:52.771 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [F:\teach\shanjupay\model1\code\generator/
12:30:52.771 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [F:\teach\shanjupay\model1\code\generator/
12:30:52.771 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [F:\teach\shanjupay\model1\code\generator/
12:30:52.771 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录： [F:\teach\shanjupay\model1\code\generator/
12:30:52.856 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/mapper.xml.ftl;  文件:F:\teach\shanju
```

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 。