mybatis_plus【条件构造】

作者：haxLook
原文链接：https://ld246.com/article/1617245261174
来源网站：链滴

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，简化开发，提高，效率而生，Mybatis-Plus是由baomidou（苞米豆）组织开发并且开源的。

官网地址

文档地址

GIT源码地址

mybatis_plus具有强大的 CRUD 操作，：内置通用 Mapper、通用 Service，仅仅通过少量配置即可现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求，也可以使用支持 Lambda 式调用。非常的简洁，在日常的工作中使用简洁的sql构造还是非常的方便。

具体的语法不在这个依次给大家介绍了，主要是在分享自己几个在工作中使用操作和注意点【下面的置都是正对于springboot中的配置】

基本配置

- mapperLocations

MyBatis Mapper 所对应的 XML 文件位置，或者是直接在yml文件中定义号xml的文件的映射地址。

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

- configLocation

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。

- typeAliasesPackage

MyBaits 别名包扫描路径，通过该属性可以给包中的类注册别名，在xml文件中可以使用包全名，也是对用的entity的路径地址

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

- idType

全局默认主键类型，设置后，即可省略实体对象中的@TableId(type = IdType.AUTO)配置。

mybatis-plus.global-config.db-config.id-type=auto

- tablePrefix

表名前缀，全局配置后可省略@TableName()配置。配置了可以省略表明配置，但是一般使用不多，为业务上复杂，表名的前缀一般都不一致。

- mapUnderscoreToCamelCase

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。栗子：【数据库字段：user_name javabea :userName】

- cacheEnabled

全局地开启或关闭配置文件中的所有映射器已经配置的任何缓存，默认为 true。

上述的配置如果是直接写在application.yml文件的话，固定的格式，如果是写在xml文件中话，可以接参照官网。

```
mybatisPlus:
  mapperLocations: classpath:/com/xxx/module/*/*/dao/xml/*.xml,classpath:/com/xxx/modul
/*/*/*/dao/xml/*.xml
  typeAliasesPackage: com.xxx.module.*.*.entity
```

条件构造器

因为条件构造器太多，在这里只说明几种说法，可以到官网上面直接查看很详细

```
allEq(Map<R, V> params)
allEq(Map<R, V> params, boolean null2IsNull)
allEq(boolean condition, Map<R, V> params, boolean null2IsNull)
allEq(BiPredicate<R, V> filter, Map<R, V> params)
allEq(BiPredicate<R, V> filter, Map<R, V> params, boolean null2IsNull)
allEq(boolean condition, BiPredicate<R, V> filter, Map<R, V> params, boolean
null2IsNull)
```

allEq使用全部结构

null2IsNull = fasle:表示的是如果map的里面又存在条件为null的情况的话，不参与sql构造

null2IsNull = true,调用的是isnull的方法

(BiPredicate filter) : **表示的是对传入参数map的过滤操作 key和vlaue的过滤操作。**

栗子:

```
例1: allEq((k,v) -> k.indexOf("s") > 0, {id:1,name:"小花",age:null}) ---> name = '老王' and sex nul


例2: allEq((k,v) -> k.indexOf("s") > 0, {id:1,name:"小花",sex:null}, false) ---> name = '小花'
```

其他的eq【等于】，le【小于】， lt【小于等于】，ge【大于】，gt【大于等于】，between 【BE WEEN 值1 AND 值2】 ，notBetween【NOT BETWEEN 值1 AND 值2】，IN【集合】，notIn【在这个集合中】，like【模糊】，notLike【NOT LIKE '%值%'】 ，likeLeft【name like '%王'】，lik Right【name like '王%'】等。。官网get

- 需要注意and 和 or 的使用

一般的条件连接都是使用的and ,如果直接直接使用or的话，表示连接的是 and a= 'a' or b= 'b'

如果想构造出 **and(a = 'a' or b = 'b')**

栗子:

```
 orderMapper.list(new QueryWrapper<Order>()
            .eq(order.BILLCODE,'1')
```

```
        .or(Wrapper->
    wrapper.eq(order.BILLCODE,'2')
        .eq(order.BILLCODE,'3'))
         .eq(order.BILLCODE,'4'));
```

这个时候表示就是上面的需求的结果。

QueryWrapper

QueryWrapper userQueryWrapper = new QueryWrapper<>();

继承自 AbstractWrapper ,自身的内部属性 entity 也用于生成 where 条件及 LambdaQueryWrapper
 可以通过 new QueryWrapper().lambda() 方法获取

UpdateWrapper

继承自 AbstractWrapper ,自身的内部属性 entity 也用于生成 where 条件及 LambdaUpdateWrapp
r, 可以通过 new UpdateWrapper().lambda() 方法获取!

可以使用如下的几种:

set

```
set(String column, Object val)
set(boolean condition, String column, Object val)
```

栗子:

- 例: set("name", "老李头")
- 例: set("name", "")--->数据库字段值变为**空字符串**
- 例: set("name", null)--->数据库字段值变为null

这个需要注意一个点，在开发中，很多开发在进行数据更新的时候，是先将数据查询出来，在根据enti
y进行设置值，来进行数据的修改的，这个时候需要注意是如果直接 entity.setName(null);会出现不
变 数据情况，如果你不使用UpdateWrapper，就需要在实体类上加上一个注解@TableField(fill = Fie
dFill.UPDATE)，说到这里就说到了@TableField注解的使用

```
TableField
1.@TableField(value = "name",exist = false)
private String name ;
exist :false //表示在进行mapper实体类映射的时候,这个字段可以不存在数据库字段映射，在开发中
般用在虚拟字段使用。

2.@TableField(value = "user_name")
private String name;

value: user_name  //表示在数据中存在字段user_name 映射到实体类的name 上面

3.@TableField(fill = FieldFill.INSERT) //插入数据时进行填充
private String password;

//多种模式适配
```

```
public enum FieldFill {
/**
* 默认不处理
*/
DEFAULT,
/**
* 插入时填充字段
*/
INSERT,
/**
* 更新时填充字段
*/
UPDATE,
/**
* 插入和更新时填充字段
*/
INSERT_UPDATE
}
```

setsql

setSql(String sql)

- 设置 SET 部分 SQL
- 例: setSql("name = '老李头'")

lambda

- 获取 LambdaWrapper

在QueryWrapper中是获取LambdaQueryWrapper

在UpdateWrapper中是获取LambdaUpdateWrapper

- 下面的栗子分别为 LambdaQueryWrapper、LambdaUpdateWrapper

```
/**
 * lambda链式编程
 */
@Test
public void test01(){
    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.lambda()
            .between(User::getAge,10,60)
            .orderByDesc(User::getId);
    List<User> list = userService.list(queryWrapper);
    list.forEach(System.out::println);
}
```

```
/**
 * 测试链式编程更新操作
 */
@Test
public void test02(){
    UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();
    updateWrapper.lambda()
            .le(User::getAge, 20)
            .setSql("email = 'haxdjf.top'");
    userService.update(updateWrapper);
}
```

Wrapper 自定义SQL

需求来源:

在使用了mybatis-plus之后, 自定义SQL的同时也想使用Wrapper的便利应该怎么办？ 在mybatis-plus本3.0.7得到了完美解决 版本需要大于或等于3.0.7, 以下两种方案取其一即可

- **Service.java**

直接使用参数的形式进行实现

```
mysqlMapper.getAll(Wrappers.<MysqlData>lambdaQuery().eq(MysqlData::getGroup, 1));
```

- **方案一** 注解方式 Mapper.java

```
@Select("select * from mysql_data ${ew.customSqlSegment}")
List<MysqlData> getAll(@Param(Constants.WRAPPER) Wrapper wrapper);
```

- **方案二** XML形式 Mapper.xml

```
<select id="getAll" resultType="MysqlData">
    SELECT * FROM mysql_data ${ew.customSqlSegment}
</select>
```

参考：mybatis_plus官网，如有侵权请联系删除!