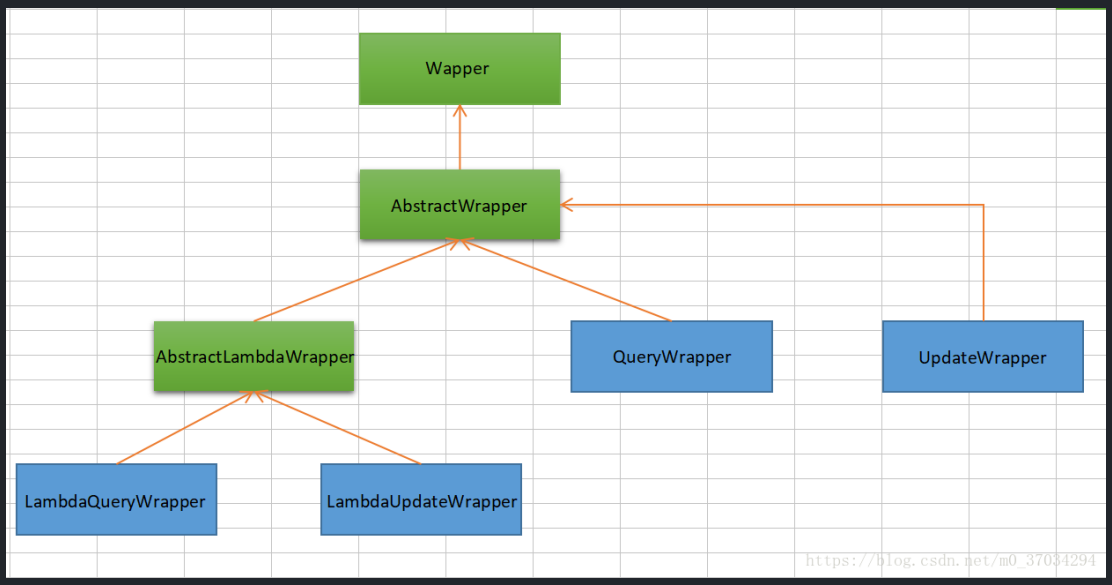
第二篇 条件构造器的应用及总结

一、条件构造器Wrapper

Mybatis Plus为我们提供了如下的一些条件构造器，我们可以利用它们实现查询条件、删除条件、更新条件的构造。



条件构造器用于给如下的Mapper方法传参，通常情况下：

- updateWrapper用于给update方法传条件参数
- queryWrapper用于给delete和select方法传参

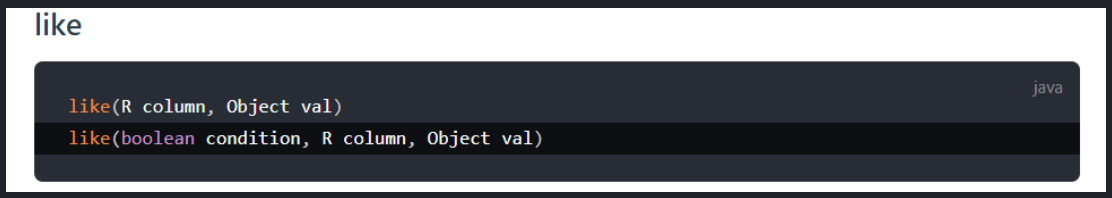
```
public interface BaseMapper<T> extends Mapper<T> {

  int delete(@Param("ew") Wrapper<T> wrapper);

  int update(@Param("et") T entity, @Param("ew")
Wrapper<T> updateWrapper);

  T selectOne(@Param("ew") Wrapper<T> queryWrapper);

  Integer selectCount(@Param("ew") Wrapper<T>
queryWrapper);
```

```
  List<T> selectList(@Param("ew") Wrapper<T>
queryWrapper);

  List<Map<String, Object>> selectMaps(@Param("ew")
Wrapper<T> queryWrapper);

  List<Object> selectObjs(@Param("ew") Wrapper<T>
queryWrapper);

  IPage<T> selectPage(IPage<T> page, @Param("ew")
Wrapper<T> queryWrapper);

  IPage<Map<String, Object>> selectMapsPage(IPage<T> page,
@Param("ew") Wrapper<T> queryWrapper);
}
```

举例一：like 条件构造

```
String name = "Jack";  //name不为空
String email = "";   //email为空串
QueryWrapper<User> query = new QueryWrapper<>();
query.like(StringUtils.isNotEmpty(name), "name", name)
      //因为email为空串，该条件未生效
     .like(StringUtils.isNotEmpty(email), "email", email);

List<User> list = userMapper.selectList(query);
list.forEach(System.out::println);
```

- QueryWrapper是查询条件构造器，like是一种条件构造函数，还有很多的条件构造函数。请参考：条件构造器



- 几乎所有的条件构造函数，都提供了condition参数实现动态SQL。也就是参数判断是否返回true，如果返回false，该条件不成立。如`email=“”`，所以`.like(StringUtils.isNotEmpty(email), "email", email);`的条件不成立。
- 所以最终的执行SQL，如下(只有name LIKE条件，没有email LIKE 条件)：

```
SELECT id,name,age,email,create_time
FROM user
WHERE name LIKE %字母%
```

举例二：allEq条件构造器

- all表示所有
- Eq是equal的缩写表示相等关系

```
//构造条件
QueryWrapper<User> query = new QueryWrapper<>();
Map<String, Object> params = new HashMap<>();
params.put("name", "Jack");
params.put("age", 18);
params.put("email", null);

// query.allEq(params,false);
query.allEq((k, v) -> !k.equals("name"), params, false);
List<User> list = userMapper.selectList(query);
list.forEach(System.out::println);
```

- 第一个参数是过滤器（可选参数），lambda表达式表示(k, v) -> !k.equals("name")，参数的Key不能是name，所以`params.put("name", "Jack");`这个查询条件被过滤掉
- 第二个参数表示传入所有的params查询参数
- 第三个参数（可选参数），表示如果值为null是否按IS NULL查询，false则忽略null列的查询，所以`params.put("email", null);`这个查询条件被过滤掉

最终执行的SQL如下：

```
SELECT id,name,age,email,create_time
FROM user
WHERE age = ?
```

更多构造器使用方法总结

请参考：官方文档：条件构造器

函数名	说明	说明/例子
eq	等于=	例: eq("name", "老王")--->name = '老王'
ne	不等于<>	例: ne("name", "老王")--->name <> '老王'
gt	大于>	例: gt("age", 18)--->age > 18
ge	大于等于>=	例: ge("age", 18)--->age >= 18
lt	小于<	例: lt("age", 18)--->age < 18
le	小于<=	例: le("age", 18)--->age <= 18
between	BETWEEN 值1 AND 值2	例: between("age", 18, 30)--->age between 18 and 30
notBetween	NOT BETWEEN 值1 AND 值2	例: notBetween("age", 18, 30)--->age not between 18 and 30
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%'
isNull	字段 IS NULL	例: isNull("name")--->name is null
isNotNull	字段 IS NOT NULL	例: isNotNull("name")--->name is not null
in	字段 IN (v0, v1, ...)	例: in("age",{1,2,3})--->age in (1,2,3)
notIn	字段 NOT IN (v0, v1, ...)	例: notIn("age", 1, 2, 3)--->age not in (1,2,3)
inSql	字段 IN (sql语句)	inSql("id", "select id from table where id < 3") --->id in (select id from table where id < 3)
notInSql	字段 NOT IN (sql语句)	notInSql("id", "select id from table where id < 3") --->age not in (select id from table where id < 3)
groupBy	分组：GROUP BY 字段, ...	例: groupBy("id", "name")--->group by id,name
orderByAsc	排序：ORDER BY 字段, ... ASC	例: orderByAsc("id", "name")--->order by id ASC,name ASC
orderByDesc	排序：ORDER BY 字段, ... DESC	例: orderByDesc("id", "name")--->order by id DESC,name DESC
orderBy	排序：ORDER BY 字段, ...	例: orderBy(true, true, "id", "name") --->order by id ASC,name ASC
having	HAVING (sql语句)	having("sum(age) > {0}", 11)--->having sum(age) > 11
or	拼接 OR	注意事项: 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) 例:eq("id",1).or().eq("name","老王")--->id = 1 or name = '老王
and	AND 嵌套	例: and(i -> i.eq("name", "李白").ne("status", "活着")) --->and (name = '李白' and status <> '活着')
apply	拼接 sql	注意事项: 该方法可用于数据库函数 动态入参的params对应前面sqlHaving内部的{index}部分.这样是不会有sql注入风险的,反之会有! 例: apply("date_format(dateColumn,'%Y-%m-%d') = {0}", "2008-08-08")--->date_format(dateColumn,'%Y-%m-%d') = '2008-08-08'")
last	无视优化规则直接拼接到 sql 的最后	无视优化规则直接拼接到 sql 的最后 注意事项: 只能调用一次,多次调用以最后一次为准 有sql注入的风险,请谨慎使用 例: last("limit 1")
exists	拼接 EXISTS (sql语句)	例: exists("select id from table where age = 1") --->exists (select id from table where age = 1)
notExists	拼接 NOT EXISTS (sql语句)	例: notExists("select id from table where age = 1") --->not exists (select id from table where age = 1)
nested	正常嵌套 不带 AND 或者 OR	正常嵌套 不带 AND 或者 OR 例: nested(i -> i.eq("name", "李白").ne("status", "活着")) --->(name = '李白' and status <> '活着')

二、lambda条件构造器

举例一：

```
// LambdaQueryWrapper<User> lambdaQ = new
QueryWrapper<User>().lambda();
// LambdaQueryWrapper<User> lambdaQ = new
LambdaQueryWrapper<>();
LambdaQueryWrapper<User> lambdaQ = Wrappers.lambdaQuery();
lambdaQ.like(User::getName, "Jack")
       .lt(User::getAge, 18);
List<User> list = userMapper.selectList(lambdaQ);
```

lambda条件构造器，最终执行SQL如下：

```
SELECT id,name,age,email,create_time
FROM user
WHERE name LIKE %字母%
AND age < 18
```

举例二：

```
List<User> list = new LambdaQueryChainWrapper<User>
(userMapper)
        .likeRight(User::getName, "Jack")
        .and(q -> q.lt(User::getAge, 40)
                   .or()
                   .isNotNull(User::getEmail)
        )
        .list();
list.forEach(System.out::println);
```

lambda条件构造器，最终执行SQL如下：

```
SELECT id,name,age,email,create_time
FROM user
WHERE name LIKE 'Jack%'
AND ( age < 18 OR email IS NOT NULL )
```