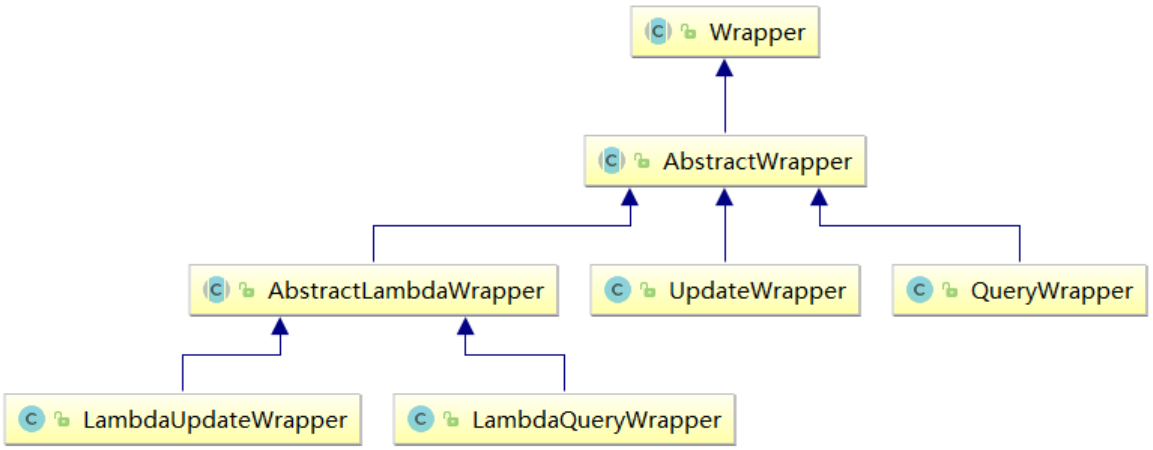
一、wapper介绍



Wrapper： 条件构造抽象类，最顶端父类

AbstractWrapper： 用于查询条件封装，生成 sql 的 where 条件

QueryWrapper： Entity 对象封装操作类，不是用lambda语法

UpdateWrapper： Update 条件封装，用于Entity对象更新操作

AbstractLambdaWrapper： Lambda 语法使用 Wrapper统一处理解析 lambda 获取 column。

LambdaQueryWrapper：看名称也能明白就是用于Lambda语法使用的查询Wrapper

LambdaUpdateWrapper： Lambda 更新封装Wrapper

二、AbstractWrapper

注意：以下条件构造器的方法入参中的 column 均表示数据库字段

1、ge、gt、le、lt、isNull、isNotNull

```
@Test
public void testSelect() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.ge("age", 28);
```

```
    List<User> users = userMapper.selectList(queryWrapper);
    System.out.println(users);
}
```

2、eq、ne

注意：seletOne返回的是一条实体记录，当出现多条时会报错

```
@Test
public void testSelectOne() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("name", "Tom");

    User user = userMapper.selectOne(queryWrapper);
    System.out.println(user);
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version FROM user WHERE deleted=0 AND name = ?

3、between、notBetween

包含大小边界

```
@Test
public void testSelectCount() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.between("age", 20, 30);

    Integer count = userMapper.selectCount(queryWrapper);
    System.out.println(count);
}
```

SELECT COUNT(1) FROM user WHERE deleted=0 AND age BETWEEN ? AND ?

4、allEq

```
@Test
public void testSelectList() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    Map<String, Object> map = new HashMap<>();
    map.put("id", 2);
    map.put("name", "Jack");
    map.put("age", 20);

    queryWrapper.allEq(map);
    List<User> users = userMapper.selectList(queryWrapper);

    users.forEach(System.out::println);
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version

FROM user WHERE deleted=0 AND name = ? AND id = ? AND age = ?

5、like、notLike、likeLeft、likeRight

selectMaps返回Map集合列表

```
@Test
public void testSelectMaps() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper
        .notLike("name", "e")
        .likeRight("email", "t");

    List<Map<String, Object>> maps = userMapper.selectMaps(queryWrapper);//返回值是Map
    maps.forEach(System.out::println);
```

```
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version

FROM user WHERE deleted=0 AND name NOT LIKE ? AND email LIKE ?

6、in、notIn、inSql、notinSql、exists、notExists

in、notIn：

- notIn("age",{1,2,3})--->age not in (1,2,3)
- notIn("age", 1, 2, 3)--->age not in (1,2,3)

inSql、notinSql：可以实现子查询

- 例: inSql("age", "1,2,3,4,5,6")--->age in (1,2,3,4,5,6)
- 例: inSql("id", "select id from table where id < 3")--->id in (select id from table where id < 3)

```
@Test
public void testSelectObjs() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    //queryWrapper.in("id", 1, 2, 3);
    queryWrapper.inSql("id", "select id from user where id < 3");

    List<Object> objects = userMapper.selectObjs(queryWrapper);//返回值是Object列表
    objects.forEach(System.out::println);
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version

FROM user WHERE deleted=0 AND id IN (select id from user where id < 3)

7、or、and

注意：这里使用的是 UpdateWrapper

不调用or则默认为使用 and 连

```java
@Test
public void testUpdate1() {

    //修改值
    User user = new User();
    user.setAge(99);
    user.setName("Andy");

    //修改条件
    UpdateWrapper<User> userUpdateWrapper = new UpdateWrapper<>();
    userUpdateWrapper
        .like("name", "h")
        .or()
        .between("age", 20, 30);

    int result = userMapper.update(user, userUpdateWrapper);

    System.out.println(result);
}
```

UPDATE user SET name=?, age=?, update_time=? WHERE deleted=0 AND name LIKE ? OR age BETWEEN ? AND ?

8、嵌套or、嵌套and

这里使用了lambda表达式，or中的表达式最后翻译成sql时会被加上圆括号

```java
@Test
public void testUpdate2() {


    //修改值
    User user = new User();
    user.setAge(99);
    user.setName("Andy");

    //修改条件
    UpdateWrapper<User> userUpdateWrapper = new UpdateWrapper<>();
```

```
    userUpdateWrapper
        .like("name", "h")
        .or(i -> i.eq("name", "李白").ne("age", 20));

    int result = userMapper.update(user, userUpdateWrapper);

    System.out.println(result);
}
```

UPDATE user SET name=?, age=?, update_time=?

WHERE deleted=0 AND name LIKE ?

OR (name = ? AND age <> ?)

9、orderBy、orderByDesc、orderByAsc

```
@Test
public void testSelectListOrderBy() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.orderByDesc("id");

    List<User> users = userMapper.selectList(queryWrapper);
    users.forEach(System.out::println);
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version

FROM user WHERE deleted=0 ORDER BY id DESC

10、last

直接拼接到 sql 的最后

注意：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

```
@Test
```

```
public void testSelectListLast() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.last("limit 1");

    List<User> users = userMapper.selectList(queryWrapper);
    users.forEach(System.out::println);
}
```

SELECT id,name,age,email,create_time,update_time,deleted,version

FROM user WHERE deleted=0 limit 1

11、指定要查询的列

```
@Test
public void testSelectListColumn() {

    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.select("id", "name", "age");

    List<User> users = userMapper.selectList(queryWrapper);
    users.forEach(System.out::println);
}
```

SELECT id,name,age FROM user WHERE deleted=0

12、set、setSql

最终的sql会合并 user.setAge()，以及 userUpdateWrapper.set() 和 setSql() 中 的字段

```
@Test
public void testUpdateSet() {

    //修改值
    User user = new User();
```

```
    user.setAge(99);

    //修改条件
    UpdateWrapper<User> userUpdateWrapper = new UpdateWrapper<>();
    userUpdateWrapper
        .like("name", "h")
        .set("name", "老李头")//除了可以查询还可以使用set设置修改的字段
        .setSql(" email = '123@qq.com'");//可以有子查询

    int result = userMapper.update(user, userUpdateWrapper);
}
```

UPDATE user SET age=?, update_time=?, name=?, email = '123@qq.com' WHERE deleted=0 AND name LIKE ?