mybatisplus

属性

@TableName("表名"),加载类前,将数据库中某表名和当前类名对应

@TableField(value = "列名"),只能影响后端到数据库中查找时候的名称,不能影响数据库返回后端时映射到对应类中属性的名称

当类中属性比数据库中字段多时: @TableField(exist = false),使用该注解,报错,但使用注解的字段为null

指定字段自动填充:在什么时机填充
列如:
1.在指定字段加上注解
@TableField(value = "create_time",fill = FieldFill.INSERT)
private Date createTime;

@TableField(value = "update_time",fill = FieldFill.INSERT_UPDATE)
private Date updateTime;

2.重写MetaObjectHandler接口,记得加上@Component注解
@Component
public class MyMetaObjectHandler implements MetaObjectHandler {

@Override
public void insertFill(MetaObject metaObject) {
this.setFieldValByName("createTime",new Date(),metaObject);
this.setFieldValByName("updateTime",new Date(),metaObject);

@Override
public void updateFill(MetaObject metaObject) {
this.setFieldValByName("updateTime",new Date(),metaObject);

最后在添加操作是指为null
User user = new User(6L, "王五", 18, "abc@gamil.com", "2",**null,null**);

@TableField(value = "id"):标识唯一主键,用于属性前
当添加数据且主键无值时,默认用雪花算法生成唯一主键值

@TableId(value = "id",type = IdType.INPUT)//指定主键生成策略
mp常见主键生成策略
NONE(1):默认的mp主键生成策略，mybatis plus会使用雪花算法(snowflake)生成19位的数值类型递增的唯一标识符TNPUT (2):要让用户自己输入主键
AUTO(0):表示使用数据表的自增主键，要求表的列是数值类型，列是递增的
ASSIGN_ID:(3):表示如果用户自己输入了id，就用用户自己输入的作为主键,如果没有指定，就用mp主键生成策略[19位]
ASSIGN_0OID (4) :表示如果用户自己输入了uuid,就用用户自己输入的uuid作为主键,如果没有指定，就用mp帮我们生成的uid作为主键

关闭驼峰映射&打印日志

mybatis-plus:
configuration:
map-underscore-to-camel-case: false//关闭驼峰映射
log-impl: org.apache.ibatis.logging.stdout.StdOutImpl//打印日志

CURD

增

User user = new User(null, "张三", 18, "abc@gamil.com", "2");
userMapper.insert(user);

删

根据id删除记录
userMapper.deleteById(6L)

批量删除
userMapper.deleteBatchIds(Arrays.asList(8, 9, 10))

简单的条件查询删除
HashMap<String, Object> map = new HashMap<>();
map.put("name", "Helen");
map.put("age", 18);
int result = userMapper.deleteByMap(map);

逻辑删除
物理删除：真实删除，将对应数据从数据库中删除，之后查询不到此条被删除数据
逻辑删除：假删除，将对应数据中代表是否被删除字段状态修改为“被删除状态”，之后在数据库中仍旧能看到此条数据记录
1.数据库中添加 deleted字段
ALTER TABLE `user` ADD COLUMN `deleted` boolean
2.实体类添加deleted 字段
@TableLogic
@TableField(fill = FieldFill.INSERT)
private Integer deleted;
3.元对象处理器接口添加deleted的insert默认值
@Override
public void insertFill(MetaObject metaObject) {
.....
this.setFieldValByName("deleted", 0, metaObject);
}
4.application.properties 加入配置
此为默认值，如果你的默认值和mp默认的一样,该配置可无
db-config:
logic-delete-field: 属性名
logic-not-delete-value: 0
logic-delete-value: 1

查

查所有
List<User> users = userMapper.selectList(null);

根据id查询记录
userMapper.selectById(1L);

通过多个id批量查询
List<User> users = userMapper.selectBatchIds(Arrays.asList(1l, 2l, 3l, 4l));
System.out.println(users.toString());

and--map
Map<String,Object> map = new HashMap<>();
map.put("字段名","lone");
map.put("字段名","18");
List<User> users = userMapper.selectByMap(map);
System.out.println(users.toString());

selectOne查询单个数据,多个会报错
userMapper.selectOne(userQueryWrapper);

selectList:查询多个或所有
List<User> users = userMapper.selectList(userQueryWrapper);//为null查询所有

统计结果个数
userMapper.selectCount(userQueryWrapper);

分页查询
0.创建配置类
@Bean
public PaginationInterceptor paginationInterceptor() {
return new PaginationInterceptor();
}

1.无条件//**这里的page和userPage是同一对象,地址相同**
Page<User> page = new Page<>(1,3);
Page<User> userPage = userMapper.selectPage(page, null);
"总记录数" +userPage.getTotal()
"总页数" +userPage.getPages()
"是否有上一页" +userPage.hasPrevious()
"是否有下一页" +userPage.hasNext()
"当前页下标" +userPage.getCurrent()

wrapper
1、
(1)eq:查询字段单个字段匹配的数据
1.QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
userQueryWrapper.eq("字段名",匹配值);
(2)eq查询多个字段匹配的数据
QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
Map<String, Object> map = new HashMap<>();
map.put("age",18);
map.put("name","lone");
userQueryWrapper.allEq(map);
(3)ne:查询不匹配字段的数据
QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
userQueryWrapper.ne("age",18);
2、
1)gt:大于等于
ge:大于
lt
le
QueryWrapper<User> queryWrapper = new QueryWrapper<>();
queryWrapper.ge("age", 28);//age>=28
3、
1)判断字段是否为空isNull、isNotNull
QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
userQueryWrapper.isNull("age");
4.包或不包含
between、notBetween
QueryWrapper<User> queryWrapper = new QueryWrapper<>();
queryWrapper.between("age", 20, 30);

5.模糊查询like、notLike、likeLeft:以什么开头、likeRight:以什么结尾
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列表
maps.forEach(System.out::println);

6.in、notIn、inSql、notInSql、exists、notExists
1)in、notIn：
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)
QueryWrapper<User> queryWrapper = new QueryWrapper<>();
//queryWrapper.in("id", 1, 2, 3);
queryWrapper.inSql("id", "select id from user where id < 3");

7、or、and
这里使用的是 UpdateWrapper
不调用or则默认为使用 and 连
UpdateWrapper<User> userUpdateWrapper = new UpdateWrapper<>();
userUpdateWrapper
.like("name", "h")
.or()
.between("age", 20, 30);
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
//修改条件
UpdateWrapper<User> userUpdateWrapper = new UpdateWrapper<>();
userUpdateWrapper
.like("name", "h")
.or(i -> i.eq("name", "李白").ne("age", 20));

9、orderBy、orderByDesc、orderByAsc
QueryWrapper<User> queryWrapper = new QueryWrapper<>();
queryWrapper.orderByDesc("id");

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

11、指定要查询的列
QueryWrapper<User> queryWrapper = new QueryWrapper<>();
queryWrapper.select("id", "name", "age");

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

改

User user = new User(6L, "王五", 20, null, null);
userMapper.updateById(user);

注意:这是根据id修改的,id不能为null,其他字段如果为null,数据库并不会对原有数据进行修改

乐观锁

当要更新一条记录的时候，希望这条记录没有被别人更新，也就是说实现线程安全的数据更新
取出记录时，获取当前version
更新时，带上这个version
执行更新时， set version = newVersion where version = oldVersion
如果version不对，就更新失败

1.数据库中添加version字段
ALTER TABLE `user` ADD COLUMN `version` INT
实体类添加version字段
开添加 @Version 注解
2、
@Version
@TableField(fill = FieldFill.INSERT)
private Integer version;

3.元对象处理器接口添加version的insert默认值
@Override
public void insertFill(MetaObject metaObject) {
.....
this.setFieldValByName("version", 1, metaObject);
}

4.在 MybatisPlusConfig 中注册 Bean
创建配置类
@Configuration
public class MybatisPlusConfig {
乐观锁插件
@Bean
public OptimisticLockerInterceptor optimisticLockerInterceptor() {
return new OptimisticLockerInterceptor();
}

测试:每次更新后version会加一

注意,执行更新语句时,必须带上数据原来的版本号,否则数据可以更新成功但是version不会加一

子主题3

特别说明:
支持的数据类型只有 int,Integer,long,Long,Date,Timestamp,LocalDateTime
整数类型下 newVersion = oldVersion + 1
newVersion 会回写到 entity 中
仅支持 updateById(id) 与 update(entity, wrapper) 方法
在 update(entity, wrapper) 方法下, wrapper 不能复用!!!