# Mybatisplus学习笔记

## @TableField

在MP中通过@TableField注解可以指定字段的一些属性，常常解决的问题有两个

1. 对象中的属性名和字段名不一致的问题（非驼峰）
2. 对象中的属性字段在表中不存在的问题

|  |
| --- |
| 1. @Data @NoArgsConstructor @AllArgsConstructor @TableName("tb\_user") public class User {  @TableId(type = IdType.*AUTO*) private Long id; private String userName; @TableField(select = false)//查询时不返回数据的值 private String password; private String name; private Integer age;  @TableField(value="email")//指定数据库中的字段名 private String mail;  @TableField(exist = false)  private String address;//在数据库表中是不存在的 } |

## 更新操作

在MP中，页面操作有两种，一种是根据id更新，另一种是根据条件更新

### 根据id更新

|  |
| --- |
| @Test public void testUpdateById(){  User user=new User();  user.setId(1L);//条件：根据id更新  user.setAge(19);//更新的字段  user.setPassword("666666");  int i = this.userMapper.updateById(user);  System.*out*.println("i"+i); } |

### 根据条件更新

|  |
| --- |
| @Test public void testUpdate(){  User user=new User();  user.setAge(20);  user.setPassword("888888");  QueryWrapper<User> queryWrapper=new QueryWrapper<>();  queryWrapper.eq("user\_name","zhangsan");//根据条件匹配user\_name=zhangsan  //根据条件更新  int update = this.userMapper.update(user, queryWrapper);  System.*out*.println("update=>"+update); } @Test public void testUpdate2(){  UpdateWrapper<User> queryWrapper=new UpdateWrapper<>();  queryWrapper.set("password","999999").set("age",401)//更新的字段  .eq("user\_name","zhangsan");//更新的条件  //根据条件更新  int update = this.userMapper.update(null, queryWrapper);  System.*out*.println("update=>"+update); } |

### 删除操作

|  |
| --- |
| @Test public void testDeleteById(){  //根据id删除我们的数据  int i = this.userMapper.deleteById(9);  System.*out*.println(i); } |

#### DeleteByMap

|  |
| --- |
| @Test public void testDeleteByMap(){  Map<String,Object> map=new HashMap<>();  map.put("user\_name","zhangsan");  map.put("password","999999");  int result=this.userMapper.deleteByMap(map);  //根据map删除数据，多条件之间是and关系  System.*out*.println(result); } |

#### Delete

|  |
| --- |
| @Test  public void testDelete(){ // //用法一： // QueryWrapper<User> wrapper=new QueryWrapper<>(); // wrapper.eq("user\_name","caocao1").eq("password","123456");  //用法二  User user=new User();  user.setPassword("123456");  user.setUserName("caocao1");  QueryWrapper<User> wrapper=new QueryWrapper<>(user);  //根据我们包装的条件进行删除  int delete = this.userMapper.delete(wrapper);  System.*out*.println("delete=>"+delete);  } |

#### DeleteBatchlds

|  |
| --- |
| @Test public void testDeleteBatchIds(){  //根据id批量删除数据  int i = this.userMapper.deleteBatchIds(Arrays.*asList*(10L, 11L));  System.*out*.println(i); } |

## 查询操作

MP提供了多种查询操作，包括根据id查询、查询单条数据、查询列表、分页查询等操作

### SelectById

|  |
| --- |
| @Test public void testSelectById(){  User user = this.userMapper.selectById(2L);  System.*out*.println(user); } |

### selectBatchlds

|  |
| --- |
| @Test public void testSelectBatchlds(){  //批量查询数据  List<User> users = this.userMapper.selectBatchIds(Arrays.*asList*(2L, 3L));  for (User user : users) {  System.*out*.println(user);  } } |

### SelectOne

|  |
| --- |
| @Test public void testSelectOne(){  QueryWrapper<User> wrapper=new QueryWrapper<>();  //设置查询条件  wrapper.eq("user\_name","lisi");  User user = this.userMapper.selectOne(wrapper);  System.*out*.println(user); } |

### selectCount

|  |
| --- |
| @Test public void testSelectCount(){  QueryWrapper<User> wrapper=new QueryWrapper<>();  wrapper.gt("age",20);//条件：年龄大于20岁的用户  //根据条件，查询数据的条数  Integer integer = this.userMapper.selectCount(wrapper);  System.*out*.println(integer); } |

### selectList

|  |
| --- |
| @Test public void testSelectList(){  QueryWrapper<User> wrapper=new QueryWrapper<>(); //设置查询条件  wrapper.like("email","itcast");  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.*out*.println(user);   } } |

### selectPage

|  |
| --- |
| //测试分页查询 @Test public void testSelectPage(){  Page<User> page=new Page<>(1,1);  QueryWrapper<User> wrapper=new QueryWrapper<>();  //设置查询条件  wrapper.like("email","itcast");  IPage<User> iPage=this.userMapper.selectPage(page,wrapper);  System.*out*.println("数据总条数："+iPage.getTotal());  System.*out*.println("数据总页数"+iPage.getPages());  System.*out*.println("当前页数"+iPage.getCurrent());  List<User> records = iPage.getRecords();  for (User record : records) {  System.*out*.println(record);  } } |

## 配置

### 基本配置

#### 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。MyBatis Configuration的具体内容请参考MyBatis官方文档

SpringBoot：

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

SpringMVC：

|  |
| --- |
| <!--这里使用MP提供的sqlSessionFactory，完成了Spring与MP的整合--> <bean id="sqlSessionFactory"  class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"/>  <property name="configLocation" value="classpath:mybatis-config.xml"></property> </bean> |

### mapperLocations

MyBatis Mapper所对应的xml文件配置，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

SpringBoot：

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

### typeAiasesPackage

MyBatis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，而不是使用全限定的类名（即XML中调用的时候不用包含包名）

SpringBoot：

|  |
| --- |
| #实体对象的扫描包 mybatis-plus.type-aliases-package=cn.itcast.mp.pojo |

## 进阶配置

### mapUnderxcoreToCamelCase

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aColumn（驼峰命名）的类似映射

注意：此属性在MyBatis中原默认值为false，在MyBatis-plus中此属性也将用于生成最终额SQL的select body

示例（SpringBoot）：

|  |
| --- |
| #关闭驼峰映射法的自动映射 mybatis-plus.configuration.map-underscore-to-camel-case=false |

### CacheEnabled

类型：boolean

默认值：true

全局地开启或关闭配置文件中的所有映射器已经设置的任何缓存，默认为true

|  |
| --- |
| #禁用缓存 mybatis-plus.configuration.cache-enabled=false |

## DB策略配置

### idType

类型：com.baomidou.mybatisplus.annotation.IdType

默认值：ID\_WORKER

全局默认主键类型，社会后，即可省略实体对象中的@Tabled（type=IdType.AUTO）配置

SpringBoot：

|  |
| --- |
| #配置全局id生成的策略 mybatis-plus.global-config.db-config.id-type=*auto* |

### tablePrefix

类型：String

默认值：null

表前缀，全局配置后可省略@TableName（）的配置

SpringBoot：

|  |
| --- |
| #全局的表名的前缀 mybatis-plus.global-config.db-config.table-prefix=tb\_ |
|  |
|  |

### allEq

|  |
| --- |
| @Test  public void testAllEq(){  Map<String,Object> params=new HashMap<>();  params.put("name","李四");  params.put("age","20");  params.put("password",null);   QueryWrapper<User> wrapper=new QueryWrapper<>(); // wrapper.allEq(params);  wrapper.allEq(params,false);  wrapper.allEq((k,v)->(k.equals("age") || k.equals("id")),params);  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.*out*.println(user);   }  } |

## 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dengyu

### Between

BETWEEN 值1 and值2

NotBETWEEM

NOT BETWEEN 值1 AND 值2

### In

字段·in（value.get(0),value.get(1)…）

### Notin

字段 NOT IN（v0，v1,….）

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

## 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%’

## 排序

### Orderby

排序：order by字段….

例：order by （true，true，“id”，“name”）🡪order by id ASC，nameASC

### OrderBYASC

排序：order by字段，。。。ASC

例如：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

|  |
| --- |
| @Test public void testOrderByAgeDesc(){  QueryWrapper<User> wrapper=new QueryWrapper<>();  //按照年龄倒序排序  wrapper.orderByDesc("name");  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.*out*.println(user);   } } |

## 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，则默认使用and连接）

### And

AND连接

例：and（i->i.eq(“name”，“李白”).ne（“status”，“活着”））🡪and（name=‘李白’ and status<>’活着’）

### 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

|  |
| --- |
| @Test  public void testSelect(){  QueryWrapper<User> wrapper=new QueryWrapper<>(); wrapper.eq("name","王五").or().eq("age",21).select("id","name","age");//指定我们查询的字段  List<User> users = this.userMapper.selectList(wrapper);  for (User user : users) {  System.*out*.println(user);    }  } |

# Mybatis-plus

## ActiveRecord

ActiveRecord的主要思想：

* 每一个数据库表对应创建一个类，类的每一个对象实例对应于数据库中表的一行记录；通常表的每一个字段都有相对应的field
* ActiveRecord同时负责把自己持久化，在ActiveRecord中封装了对数据库的访问，即CURD
* ActiveRecord是一种领域模型（Domain Model）,封装了部分业务逻辑

|  |
| --- |
| @Test public void testSelectById(){  User user=new User();  user.setId(2L);  User user1 = user.selectById();  System.*out*.println(user1); } |