

学习目标

- ☐ 能够基于MyBatisPlus完成标准Dao开发
- ☐ 能够掌握MyBatisPlus的条件查询
- ☐ 能够掌握MyBatisPlus的字段映射与表名映射
- ☐ 能够掌握id生成策略控制
- ☐ 能够理解代码生成器的相关配置

一、MyBatisPlus简介

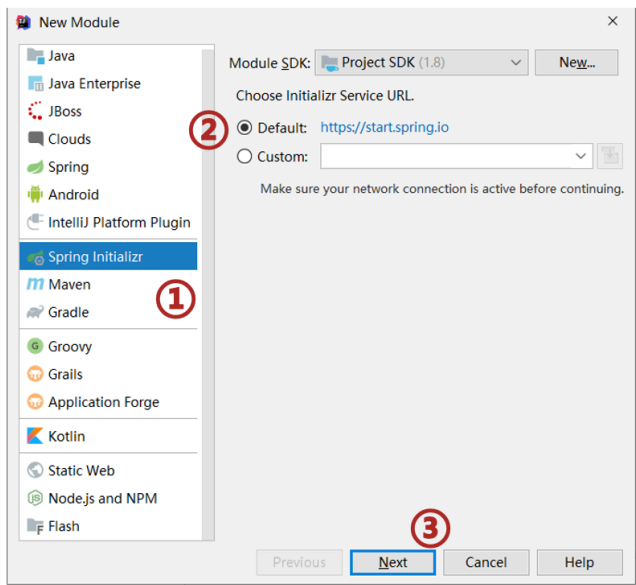
1. 入门案例

问题导入

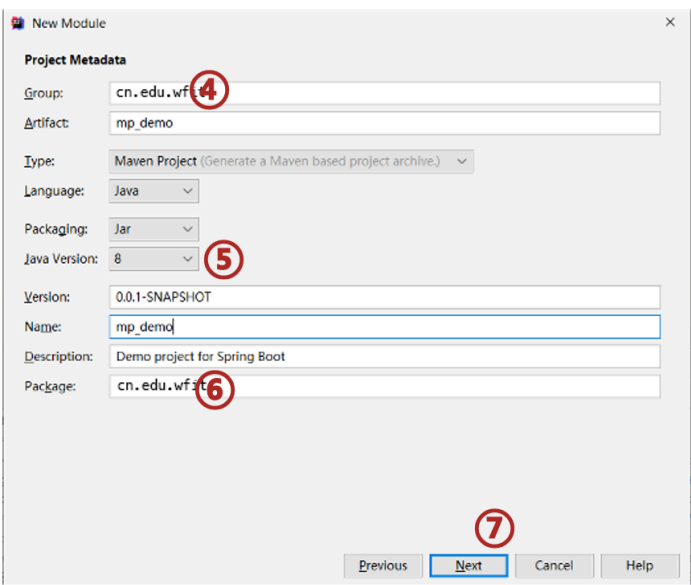
MyBatisPlus环境搭建的步骤？

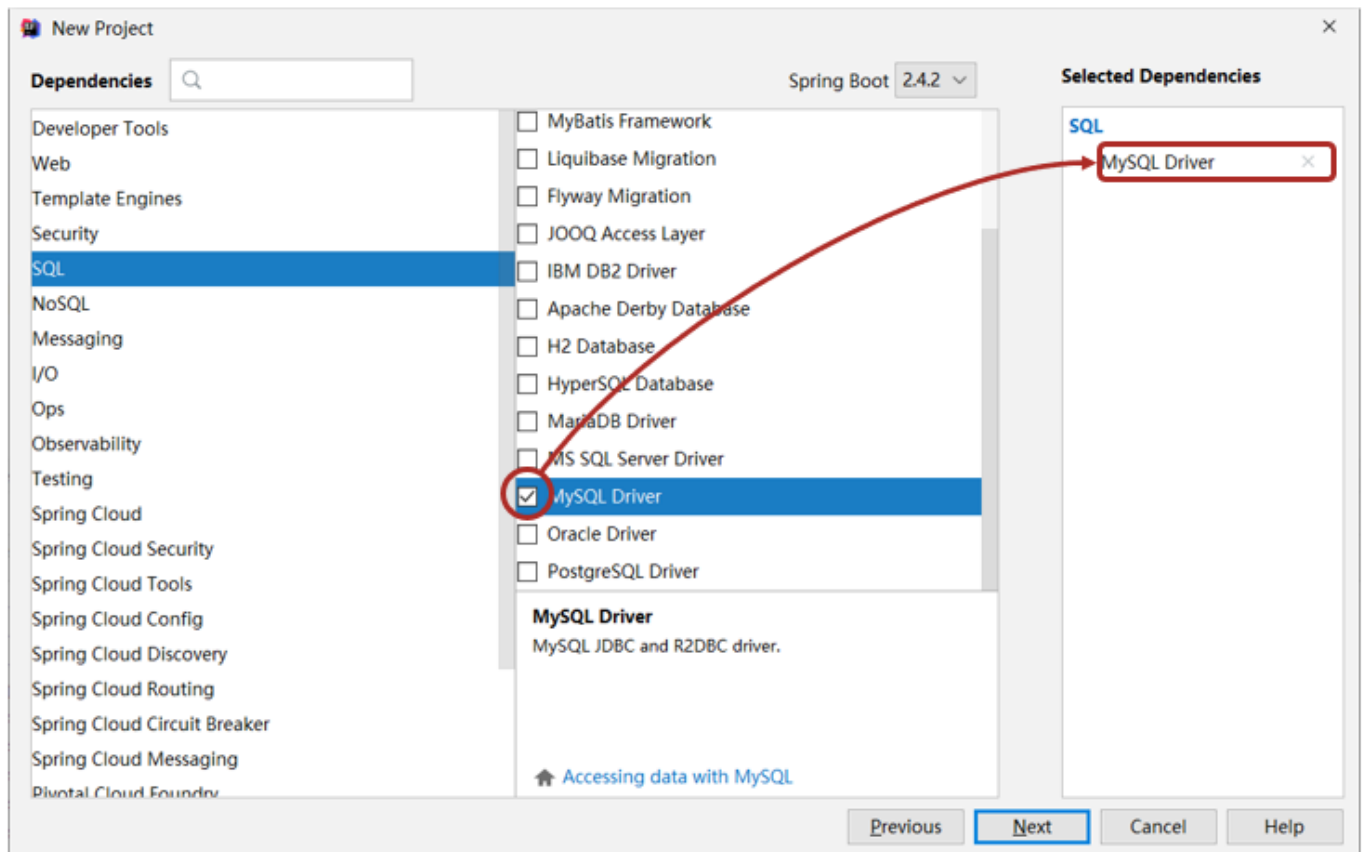
1.1 SpringBoot整合MyBatisPlus入门程序

①：创建新模块，选择Spring初始化，并配置模块相关基础信息



②：选择当前模块需要使用的技术集（仅保留JDBC）





③：手动添加MyBatisPlus起步依赖

```
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.5.3.1</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid-spring-boot-starter</artifactId>
  <version>1.2.18</version>
</dependency>
```

注意事项1：由于mp并未被收录到idea的系统内置配置，无法直接选择加入

注意事项2：如果使用Druid数据源，需要导入对应坐标

④：制作实体类与表结构

(类名与表名对应，属性名与字段名对应)

```
create database if not exists mybatisplus_db character set utf8;
use mybatisplus_db;
CREATE TABLE user (
  id bigint(20) primary key auto_increment,
  name varchar(32) not null,
  password varchar(32) not null,
```

```

        age int(3) not null ,
        tel varchar(32) not null
    );
insert into user values(null,'tom','123456',12,'12345678910');
insert into user values(null,'jack','123456',8,'12345678910');
insert into user values(null,'jerry','123456',15,'12345678910');
insert into user values(null,'tom','123456',9,'12345678910');
insert into user values(null,'snake','123456',28,'12345678910');
insert into user values(null,'张益达','123456',22,'12345678910');
insert into user values(null,'张大炮','123456',16,'12345678910');

```

```

public class User {
    private Long id;
    private String name;
    private String password;
    private Integer age;
    private String tel;
    //自行添加getter、setter、toString()等方法
}

```

⑤：设置Jdbc参数 (application.yml)

```

spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mybatisplus_db?serverTimezone=UTC
    username: root
    password: root

```

⑥：定义数据接口，继承BaseMapper

```

package cn.edu.wfit.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import cn.edu.wfit.User;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface UserDao extends BaseMapper<User> {
}

```

⑦：测试类中注入dao接口，测试功能

```
package cn.edu.wfit;

import cn.edu.wfit.dao.UserDao;
import cn.edu.wfit.domain.User;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

@SpringBootTest
public class Mybatisplus01QuickstartApplicationTests {

    @Autowired
    private UserDao userDao;

    @Test
    void testGetAll() {
        List<User> userList = userDao.selectList(null);
        System.out.println(userList);
    }
}
```

2. MyBatisPlus概述

问题导入

通过入门案例制作，MyBatisPlus的优点有哪些？

2.1 MyBatis介绍

- MyBatisPlus（简称MP）是基于MyBatis框架基础上开发的增强型工具，旨在简化开发、提高效率
- 官网：<https://mybatis.plus/> <https://mp.baomidou.com/>

2.2 MyBatisPlus特性

- 无侵入：只做增强不做改变，不会对现有工程产生影响
- 强大的 CRUD 操作：内置通用 Mapper，少量配置即可实现单表CRUD 操作
- 支持 Lambda：编写查询条件无需担心字段写错
- 支持主键自动生成
- 内置分页插件
-

二、标准数据层开发

1. MyBatisPlus的CRUD操作

功能	自定义接口	MP接口
新增	<small>之前我们自定义的dao/mapper接口方法</small> <code>boolean save(T t)</code>	<code>int insert(T t)</code>
删除	<code>boolean delete(int id)</code>	<code>int deleteById(Serializable id)</code>
修改	<code>boolean update(T t)</code>	<code>int updateById(T t)</code>
根据id查询	<code>T getById(int id)</code>	<code>T selectById(Serializable id)</code>
查询全部	<code>List<T> getAll()</code>	<code>List<T> selectList()</code>
分页查询	<code>PageInfo<T> getAll(int page, int size)</code>	<code>IPage<T> selectPage(IPage<T> page)</code>
按条件查询	<code>List<T> getAll(Condition condition)</code>	<code>IPage<T> selectPage(Wrapper<T> queryWrapper)</code>

```
package cn.edu.wfit;

import cn.edu.wfit.dao.UserDao;
import cn.edu.wfit.domain.User;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

@SpringBootTest
class Mybatisplus01QuickstartApplicationTests {

    @Autowired
    private UserDao userDao;

    @Test
    void testSave() {
        User user = new User();
        user.setName("张三");
        user.setPassword("123456");
        user.setAge(12);
        user.setTel("13863600158");
        userDao.insert(user);
    }

    @Test
    void testDelete() {
        userDao.deleteById(1401856123725713409L);
    }

    @Test
```

```

void testUpdate() {
    User user = new User();
    user.setId(1L);
    user.setName("Tom888");
    user.setPassword("tom888");
    userDao.updateById(user);
}

@Test
void testGetById() {
    User user = userDao.selectById(2L);
    System.out.println(user);
}

@Test
void testGetAll() {
    List<User> userList = userDao.selectList(null);
    System.out.println(userList);
}
}

```

2. Lombok插件介绍

问题导入

有什么简单的办法可以自动生成实体类的GET、SET方法？

- Lombok，一个Java类库，提供了一组注解，简化POJO实体类开发。

```

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
</dependency>

```

- 常用注解：==@Data==，为当前实体类在编译期设置对应的get/set方法，无参/无参构造方法，toString方法，hashCode方法，equals方法等

```

package com.itheima.domain;

import lombok.*;

/*
    1 生成getter和setter方法：@Getter、@Setter
    生成toString方法：@ToString
    生成equals和hashCode方法：@EqualsAndHashCode

    2 统一成以上所有：@Data

```

3 生成空参构造: @NoArgsConstructor
生成全参构造: @AllArgsConstructor

4 lombok还给我们提供了builder的方式创建对象,好处就是可以链式编程。 @Builder 【扩展】

```
*/  
@Data  
public class User {  
    private Long id;  
    private String name;  
    private String password;  
    private Integer age;  
    private String tel;  
}
```

3. MyBatisPlus分页功能

问题导入

思考一下Mybatis分页插件是如何用的?

3.1 分页功能接口

功能	自定义接口	MP接口
新增	<code>boolean save(T t)</code>	<code>int insert(T t)</code>
删除	<code>boolean delete(int id)</code>	<code>int deleteById(Serializable id)</code>
修改	<code>boolean update(T t)</code>	<code>int updateById(T t)</code>
根据id查询	<code>T getById(int id)</code>	<code>T selectById(Serializable id)</code>
查询全部	<code>List<T> getAll()</code>	<code>List<T> selectList()</code>
分页查询	<code>PageInfo<T> getAll(int page, int size)</code>	<code>IPage<T> selectPage(IPage<T> page)</code>
按条件查询	<code>List<T> getAll(Condition condition)</code>	<code>IPage<T> selectPage(Wrapper<T> queryWrapper)</code>

3.2 MyBatisPlus分页使用

①: 设置分页拦截器作为Spring管理的bean

```
package cn.edu.wfit.config;  
  
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;  
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;
```

```

@Configuration
public class MybatisPlusConfig {

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        //1 创建MybatisPlusInterceptor拦截器对象
        MybatisPlusInterceptor mpInterceptor=new MybatisPlusInterceptor();
        //2 添加分页拦截器
        mpInterceptor.addInnerInterceptor(new PaginationInnerInterceptor());
        return mpInterceptor;
    }
}

```

②：执行分页查询

```

//分页查询
@Test
void testSelectPage(){
    //1 创建IPage分页对象,设置分页参数
    IPage<User> page=new Page<>(1,3);
    //2 执行分页查询
    userDao.selectPage(page,null);
    //3 获取分页结果
    System.out.println("当前页码值: "+page.getCurrent());
    System.out.println("每页显示数: "+page.getSize());
    System.out.println("总页数: "+page.getPages());
    System.out.println("总条数: "+page.getTotal());
    System.out.println("当前页数据: "+page.getRecords());
}

```

3.3 开启MyBatisPlus日志

```

spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mybatisplus_db?serverTimezone=UTC
    username: root
    password: root
# 开启mp的日志（输出到控制台）
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdoutImpl

```


3.4 解决日志打印过多问题

3.4.1 取消初始化spring日志打印

```
✓ Tests passed: 1 of 1 test – 455 ms
15:35:40.243 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper
15:35:40.248 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @Cont
15:35:40.248 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default
15:35:40.248 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default
15:35:40.248 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default
15:35:40.248 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not
```

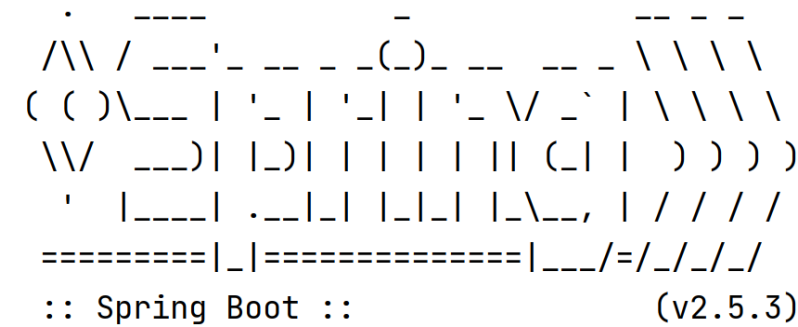
做法：在resources下新建一个logback.xml文件，名称固定，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

</configuration>
```

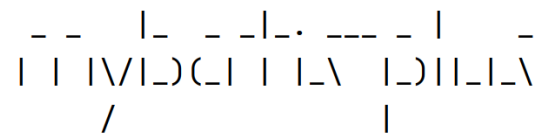
关于logback参考播客：<https://www.jianshu.com/p/75f9d11ae011>

3.4.2 取消SpringBoot启动banner图标



```
spring:
  main:
    banner-mode: off # 关闭SpringBoot启动图标(banner)
```

3.4.3 取消MybatisPlus启动banner图标













```
# mybatis-plus日志控制台输出
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdoutImpl
  global-config:
    banner: off # 关闭mybatisplus启动图标
```

三、DQL编程控制

1. 条件查询方式

- MyBatisPlus将书写复杂的SQL查询条件进行了封装，使用编程的形式完成查询条件的组合

```
(m)  selectById(Serializable): T
(m)  selectBatchIds(Collection<? extends Serializable>): List<T>
(m)  selectByMap(Map<String, Object>): List<T>
(m)  selectOne(Wrapper<T>): T
(m)  selectCount(Wrapper<T>): Integer
(m)  selectList(Wrapper<T>): List<T>
(m)  selectMaps(Wrapper<T>): List<Map<String, Object>>
(m)  selectObjs(Wrapper<T>): List<Object>
(m)  selectPage(IPage<T>, Wrapper<T>): IPage<T>
(m)  selectMapsPage(IPage<T>, Wrapper<T>): IPage<Map<String, Object>>
```

1.1 条件查询

1.1.1 方式一：按条件查询

```
//方式一：按条件查询
QueryWrapper<User> qw=new QueryWrapper<>();
qw.lt("age", 18);
List<User> userList = userDao.selectList(qw);
System.out.println(userList);
```

1.1.2 方式二：lambda格式按条件查询

```
//方式二：lambda格式按条件查询
QueryWrapper<User> qw = new QueryWrapper<User>();
qw.lambda().lt(User::getAge, 10);
List<User> userList = userDao.selectList(qw);
System.out.println(userList);
```

1.1.3 方式三：lambda格式按条件查询（推荐）

//方式三：lambda格式按条件查询

```
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();  
lqw.lt(User::getAge, 10);  
List<User> userList = userDao.selectList(lqw);  
System.out.println(userList);
```

1.2 组合条件

1.2.1 并且关系 (and)

//并且关系

```
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();  
//并且关系：10到30岁之间  
lqw.lt(User::getAge, 30).gt(User::getAge, 10);  
List<User> userList = userDao.selectList(lqw);  
System.out.println(userList);
```

1.2.2 或者关系 (or)

//或者关系

```
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();  
//或者关系：小于10岁或者大于30岁  
lqw.lt(User::getAge, 10).or().gt(User::getAge, 30);  
List<User> userList = userDao.selectList(lqw);  
System.out.println(userList);
```

1.3 NULL值处理

问题导入

如下搜索场景，在多条件查询中，有条件的值为空应该怎么解决？

The screenshot shows a search interface with various filters. Red arrows point to three empty input fields: the '收货地' (Delivery Address) dropdown menu, and two price range inputs labeled '¥ 请输入'.

电器城首页	手机馆	苏宁易购
全部 > 手机		
品牌	Apple, 小米, HUAWEI, SAMSUNG, 360, Lenovo	
综合	人气	新品
销量	价格	收货地: 杭州
¥ 请输入		¥ 请输入
新到商品		包邮
折扣		更多
店铺		大图

有部分筛选条件没有值

1.3.1 if语句控制条件追加

```
Integer minAge=10; //将来有用户传递进来,此处简化成直接定义变量了
Integer maxAge=null; //将来有用户传递进来,此处简化成直接定义变量了
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
if(minAge!=null){
    lqw.gt(User::getAge, minAge);
}
if(maxAge!=null){
    lqw.lt(User::getAge, maxAge);
}
List<User> userList = userDao.selectList(lqw);
userList.forEach(System.out::println);
```

1.3.2 条件参数控制

```
Integer minAge=10; //将来有用户传递进来,此处简化成直接定义变量了
Integer maxAge=null; //将来有用户传递进来,此处简化成直接定义变量了
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//参数1: 如果表达式为true, 那么查询才使用该条件
lqw.gt(minAge!=null, User::getAge, minAge);
lqw.lt(maxAge!=null, User::getAge, maxAge);
List<User> userList = userDao.selectList(lqw);
userList.forEach(System.out::println);
```

1.3.3 条件参数控制（链式编程）

```
Integer minAge=10; //将来有用户传递进来,此处简化成直接定义变量了
Integer maxAge=null; //将来有用户传递进来,此处简化成直接定义变量了
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//参数1: 如果表达式为true, 那么查询才使用该条件
lqw.gt(minAge!=null, User::getAge, minAge)
    .lt(maxAge!=null, User::getAge, maxAge);
List<User> userList = userDao.selectList(lqw);
userList.forEach(System.out::println);
```

2. 查询投影-设置【查询字段、分组、分页】

2.1 查询结果包含模型类中部分属性

```

/*LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
lqw.select(User::getId, User::getName, User::getAge);*/
//或者
QueryWrapper<User> lqw = new QueryWrapper<User>();
lqw.select("id", "name", "age", "tel");
List<User> userList = userDao.selectList(lqw);
System.out.println(userList);

```

2.2 查询结果包含模型类中未定义的属性

```

QueryWrapper<User> lqw = new QueryWrapper<User>();
lqw.select("count(*) as count, tel");
lqw.groupBy("tel");
List<Map<String, Object>> userList = userDao.selectMaps(lqw);
System.out.println(userList);

```

3. 查询条件设定

问题导入

多条件查询有哪些组合？

- 范围匹配 (>、=、between)
- 模糊匹配 (like)
- 空判定 (null)
- 包含性匹配 (in)
- 分组 (group)
- 排序 (order)
-

3.1 查询条件

- 用户登录 (eq匹配)

```

LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//等同于=
lqw.eq(User::getName, "Jerry").eq(User::getPassword, "jerry");
User loginUser = userDao.selectOne(lqw);
System.out.println(loginUser);

```

- 购物设定价格区间、户籍设定年龄区间 (le ge匹配 或 between匹配)

```

LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//范围查询 lt le gt ge eq between
lqw.between(User::getAge, 10, 30);
List<User> userList = userDao.selectList(lqw);
System.out.println(userList);

```

- 查信息，搜索新闻（非全文检索版：like匹配）

```
LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//模糊匹配 like
lqw.likeLeft(User::getName, "J");
List<User> userList = userDao.selectList(lqw);
System.out.println(userList);
```

- 统计报表（分组查询聚合函数）

```
QueryWrapper<User> qw = new QueryWrapper<User>();
qw.select("gender", "count(*) as nums");
qw.groupBy("gender");
List<Map<String, Object>> maps = userDao.selectMaps(qw);
System.out.println(maps);
```

3.2 查询API

- 更多查询条件设置参看 <https://mybatis.plus/guide/wrapper.html#abstractwrapper>

3.3 练习：MyBatisPlus练习

题目：基于MyBatisPlus_Ex1模块，完成Top5功能的开发。

- 说明：
 - ①：Top5指根据销售量排序（提示：对销售量进行降序排序）
 - ②：Top5是仅获取前5条数据（提示：使用分页功能控制数据显示数量）

4. 字段映射与表名映射

问题导入

思考表的字段和实体类的属性不对应，查询会怎么样？

4.1 问题一：表字段与编码属性设计不同步

- 在模型类属性上方，使用@TableField属性注解，通过==value==属性，设置当前属性对应的数据库表中的字段关系。

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(32),
  `pwd` varchar(32),
  `age` int(3),
  `tel` varchar(32),
  PRIMARY KEY (`id`)
)
```

```
public class User {
  private Long id;
  private String name;
  @TableField(value="pwd")
  private String password;
  private Integer age;
  private String tel;
}
```

我改

4.2 问题二：编码中添加了数据库中未定义的属性

- 在模型类属性上方，使用@TableField注解，通过==exist==属性，设置属性在数据库表字段中是否存在，默认为true。此属性无法与value合并使用。

```
CREATE TABLE `user` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `name` varchar(32),  
  `pwd` varchar(32),  
  `age` int(3),  
  `tel` varchar(32),  
  PRIMARY KEY (`id`)  
)
```

```
public class User {  
    private Long id;  
    private String name;  
    @TableField(value="pwd")  
    private String password;  
    private Integer age;  
    private String tel;  
    @TableField(exist = false)  
    private Integer online;  
}
```

该属性在数据库字段中不存在

4.3 问题三：采用默认查询开放了更多的字段查看权限

- 在模型类属性上方，使用@TableField注解，通过==select==属性：设置该属性是否参与查询。此属性与select()映射配置不冲突。

```
SELECT id, name, pwd, age, tel, speciality FROM user
```

```
CREATE TABLE `user` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `name` varchar(32),  
  `pwd` varchar(32),  
  `age` int(3),  
  `tel` varchar(32),  
  PRIMARY KEY (`id`)  
)
```

```
public class User {  
    private Long id;  
    private String name;  
    @TableField(value="pwd", select = false)  
    private String password;  
    private Integer age;  
    private String tel;  
    @TableField(exist = false)  
    private Integer online;  
}
```

4.4 问题四：表名与编码开发设计不同步

- 在模型类上方，使用@TableName注解，通过==value==属性，设置当前类对应的数据库表名称。

```
CREATE TABLE `tbl_user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(32),
  `pwd` varchar(32),
  `age` int(3),
  `tel` varchar(32),
  PRIMARY KEY (`id`)
)
```

```
@TableName("tbl_user")
public class User {
    private Long id;
    private String name;
    @TableField(value="pwd",select = false)
    private String password;
    private Integer age;
    private String tel;
    @TableField(exist = false)
    private Integer online;
}
```

模型类对应的表名称

```
@Data
@TableName("tbl_user")
public class User {
    /*
     * id为Long类型，因为数据库中id为bigint类型，
     * 并且mybatis有自己的一套id生成方案，生成出来的id必须是Long类型
     */
    private Long id;
    private String name;
    @TableField(value = "pwd",select = false)
    private String password;
    private Integer age;
    private String tel;
    @TableField(exist = false) //表示online字段不参与CRUD操作
    private Boolean online;
}
```

四、DML编程控制

1. id生成策略控制（Insert）

问题导入

主键生成的策略有哪几种方式？

不同的表应用不同的id生成策略

- 日志：自增（1,2,3,4,）
- 购物订单：特殊规则（FQ23948AK3843）
- 外卖单：关联地区日期等信息（10 04 20200314 34 91）
- 关系表：可省略id
-

1.1 id生成策略控制（@TableId注解）

- 名称：@TableId
- 类型：属性注解
- 位置：模型类中用于表示主键的属性定义上方
- 作用：设置当前类中主键属性的生成策略
- 相关属性

type：设置主键属性的生成策略，值参照IdType枚举值

```
public class User {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

- AUTO(0)：使用数据库id自增策略控制id生成
- NONE(1)：不设置id生成策略
- INPUT(2)：用户手工输入id
- ASSIGN_ID(3)：雪花算法生成id（可兼容数值型与字符串型）
- ASSIGN_UUID(4)：以UUID生成算法作为id生成策略

1.2 全局策略配置

```
mybatis-plus:  
  global-config:  
    db-config:  
      id-type: assign_id  
      table-prefix: tbl_
```

id生成策略全局配置

```
@TableName("tbl_user")  
public class User {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

```
@TableName("tbl_score")  
public class Score {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

```
@TableName("tbl_subject")  
public class Subject {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

```
@TableName("tbl_order")  
public class Order {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

```
@TableName("tbl_equipment")  
public class Equipment {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

```
@TableName("tbl_log")  
public class Log {  
    @TableId(type = IdType.AUTO)  
    private Long id;  
}
```

表名前缀全局配置

```
@TableName("tbl_user")
public class User {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

```
@TableName("tbl_score")
public class Score {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

```
@TableName("tbl_subject")
public class Subject {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

```
@TableName("tbl_order")
public class Order {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

```
@TableName("tbl_equipment")
public class Equipment {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

```
@TableName("tbl_log")
public class Log {
    @TableId(type = IdType.AUTO)
    private Long id;
}
```

2. 多记录操作（批量Delete/Select）

问题导入

MyBatisPlus是否支持批量操作？



2.1 按照主键删除多条记录

```
//删除指定多条数据
List<Long> list = new ArrayList<>();
list.add(1402551342481838081L);
list.add(1402553134049501186L);
list.add(1402553619611430913L);

userDao.deleteBatchIds(list);
```

2.2 根据主键查询多条记录

```
//查询指定多条数据
List<Long> list = new ArrayList<>();
list.add(1L);
list.add(3L);
list.add(4L);
userDao.selectBatchIds(list);
```

3. 逻辑删除 (Delete/Update)

问题导入

在实际环境中，如果想删除一条数据，是否会真的从数据库中删除该条数据？

- 删除操作业务问题：业务数据从数据库中丢弃
- 逻辑删除：为数据设置是否可用状态字段，删除时设置状态字段为不可用状态，数据保留在数据库中

合同编号	成交日期	金额	员工编号	员工编号	姓名	工号	deleted
1	2025/4/1	100,000.00	1	1	张业绩	9501	1
2	2025/5/12	300,000.00	1	2	李小白	9502	0
3	2025/9/11	500,000.00	1	3	王笑笑	9503	0
4	2025/11/14	80,000.00	2				
5	2025/12/25	20,000.00	3				

编号	姓名	业绩
1	张业绩	900,000.00
2	李小白	80,000.00
3	王笑笑	20,000.00
合计		1,000,000.00

3.1 逻辑删除案例

①：数据库表中添加逻辑删除标记字段

字段	索引	外键	触发器	选项	注释	SQL 预览				
名			类型	长度	小数点	不是 null	虚拟	键		
id			bigint	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1		
name			varchar	32	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
pwd			varchar	32	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
age			int	3	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
tel			varchar	32	0	<input type="checkbox"/>	<input type="checkbox"/>			
* deleted			int	1		<input type="checkbox"/>	<input type="checkbox"/>			

②：实体类中添加对应字段，并设定当前字段为逻辑删除标记字段

```
package cn.edu.wfit.domain;

import com.baomidou.mybatisplus.annotation.*;

import lombok.Data;

@Data
public class User {

    private Long id;

    //逻辑删除字段，标记当前记录是否被删除
    @TableLogic
    private Integer deleted;

}
```

③：配置逻辑删除字面值

```
mybatis-plus:
  global-config:
    db-config:
      table-prefix: tbl_
      # 逻辑删除字段名
      logic-delete-field: deleted
      # 逻辑删除字面值：未删除为0
      logic-not-delete-value: 0
      # 逻辑删除字面值：删除为1
      logic-delete-value: 1
```

逻辑删除本质：逻辑删除的本质其实是修改操作。如果加了逻辑删除字段，查询数据时也会自动带上逻辑删除字段。

执行SQL语句： `UPDATE tbl_user SET deleted=1 WHERE id=? AND deleted=0`

执行数据结果：

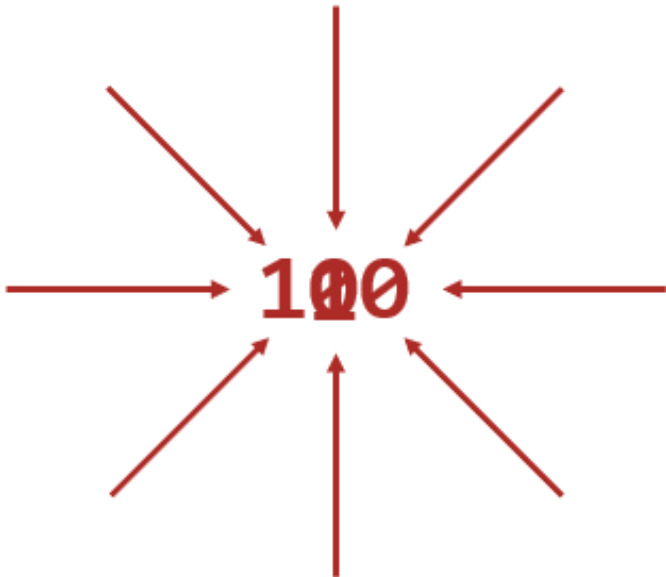
id	name	pwd	age	tel	deleted
1	Tom888	tom888	3	18866668888	1
2	Jerry	jerry	4	16688886666	0
3	Jock	123456	41	18812345678	0
4	传智播客	itcast	15	4006184000	0
5	黑马程序员	itheima	12	4006184000	0
666	黑马程序员	itheima	12	4006184000	0
667	黑马程序员	itheima	12	4006184000	0

4. 乐观锁 (Update)

问题导入

乐观锁主张的思想是什么？

- 业务并发现象带来的问题：秒杀



4.1 乐观锁案例

①：数据库表中添加锁标记字段

字段	索引	外键	触发器	选项	注释	SQL 预览					
名			类型	长度	小数点	不是 null	虚拟	键			
id			bigint	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	🔑 1			
name			varchar	32	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
pwd			varchar	32	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
age			int	3	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
tel			varchar	32	0	<input type="checkbox"/>	<input type="checkbox"/>				
deleted			int	1	0	<input type="checkbox"/>	<input type="checkbox"/>				
version			int	11	0	<input type="checkbox"/>	<input type="checkbox"/>				

②：实体类中添加对应字段，并设定当前字段为逻辑删除标记字段

```
package cn.edu.wfit.domain;

import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableLogic;
import com.baomidou.mybatisplus.annotation.Version;
import lombok.Data;

@Data
public class User {
```

```

private Long id;

@Version
private Integer version;
}

```

③：配置乐观锁拦截器实现锁机制对应的动态SQL语句拼装

```

package cn.edu.wfit.config;

import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.OptimisticLockerInnerInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MpConfig {
    @Bean
    public MybatisPlusInterceptor mpInterceptor() {
        //1.定义Mp拦截器
        MybatisPlusInterceptor mpInterceptor = new MybatisPlusInterceptor();

        //2.添加乐观锁拦截器
        mpInterceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());

        return mpInterceptor;
    }
}

```

④：使用乐观锁机制在修改前必须先获取到对应数据的version方可正常进行

```

@Test
public void testUpdate() {
    /*User user = new User();
    user.setId(3L);
    user.setName("Jock666");
    user.setVersion(1);
    userDao.updateById(user);*/

    //1.先通过要修改的数据id将当前数据查询出来
    //User user = userDao.selectById(3L);
    //2.将要修改的属性逐一设置进去
    //user.setName("Jock888");
}

```

```
//userDao.updateById(user);

//1.先通过要修改的数据id将当前数据查询出来
User user = userDao.selectById(3L); //version=3
User user2 = userDao.selectById(3L); //version=3
user2.setName("Jock aaa");
userDao.updateById(user2); //version=>4
user.setName("Jock bbb");
userDao.updateById(user); //version=3?条件还成立吗?
}
```

执行修改前先执行查询语句：

```
SELECT id,name,age,tel,deleted,version FROM tbl_user WHERE id=?
```

执行修改时使用version字段作为乐观锁检查依据

```
UPDATE tbl_user SET name=?, age=?, tel=?, version=? WHERE id=? AND version=?
```

五、快速开发-代码生成器

问题导入

如果只给一张表的字段信息，能够推演出Domain、Dao层的代码？

1. MyBatisPlus提供模板

- Mapper接口模板

造句： 警察叔叔 正 指挥交通 呢。

```
package cn.edu.wfit.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import cn.edu.wfit.domain.Book;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface BookDao extends BaseMapper<Book> {
}
```

参数

模板

- 实体对象类模板

模板

```
import com.baomidou.mybatisplus.annotation.*;
import lombok.Data;
```

```
@Data
```

```
@TableName("____")
```

```
public class User {
```

```
    @TableId(type = IdType.____)
```

```
    private Long ____;
```

```
    private String ____;
```

```
    @TableField(value = "pwd", select = false)
```

```
    private String ____;
```

```
    private Integer ____;
```

```
    private String ____;
```

```
    @TableField(exist = false)
```

```
    private Integer online;
```

```
    @TableLogic(value = "____", delval = "____")
```

```
    private Integer ____;
```

```
    @Version
```

```
    private Integer ____;
```

```
}
```

数据库读取
开发者配置

2. 工程搭建和基本代码编写

- 第一步：创建SpringBoot工程，添加代码生成器相关依赖，其他依赖自行添加

```
<!--代码生成器-->
```

```
<dependency>
```

```
    <groupId>com.baomidou</groupId>
```

```
    <artifactId>mybatis-plus-generator</artifactId>
```

```
    <version>3.4.1</version>
```

```
</dependency>
```

```
<!--velocity模板引擎-->
```

```
<dependency>
```

```
    <groupId>org.apache.velocity</groupId>
```

```
    <artifactId>velocity-engine-core</artifactId>
```

```
    <version>2.3</version>
```

```
</dependency>
```

- 第二步：编写代码生成器类

```
package cn.edu.wfit;
```

```
import com.baomidou.mybatisplus.generator.AutoGenerator;
```

```
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
```

```
public class Generator {
```

```
    public static void main(String[] args) {
```

```
        //1. 创建代码生成器对象，执行生成代码操作
```

```
        AutoGenerator autoGenerator = new AutoGenerator();
```



```

//2. 数据源相关配置：读取数据库中的信息，根据数据库表结构生成代码
DataSourceConfig dataSource = new DataSourceConfig();
dataSource.setDriverName("com.mysql.cj.jdbc.Driver");
dataSource.setUrl("jdbc:mysql://localhost:3306/mybatisplus_db?
serverTimezone=UTC");
dataSource.setUsername("root");
dataSource.setPassword("root");
autoGenerator.setDataSource(dataSource);

//3. 执行生成操作
autoGenerator.execute();
}
}

```

3. 开发者自定义配置

- 设置全局配置

```

//设置全局配置
GlobalConfig globalConfig = new GlobalConfig();
globalConfig.setOutputDir(System.getProperty("user.dir")+"/mybatisplus_04_generator/src
/main/java");    //设置代码生成位置
globalConfig.setOpen(false);    //设置生成完毕后是否打开生成代码所在的目录
globalConfig.setAuthor("张三");    //设置作者
globalConfig.setFileOverride(true);    //设置是否覆盖原始生成的文件
globalConfig.setMapperName("%sDao");    //设置数据层接口名，%s为占位符，指代模块名称
globalConfig.setIdType(IdType.ASSIGN_ID);    //设置Id生成策略
autoGenerator.setGlobalConfig(globalConfig);

```

- 设置包名相关配置

```

//设置包名相关配置
PackageConfig packageInfo = new PackageConfig();
packageInfo.setParent("com.aaa");    //设置生成的包名，与代码所在位置不冲突，二者叠加组成完整路径
packageInfo.setEntity("domain");    //设置实体类包名
packageInfo.setMapper("dao");    //设置数据层包名
autoGenerator.setPackageInfo(packageInfo);

```

- 策略设置

//策略设置

```
StrategyConfig strategyConfig = new StrategyConfig();
strategyConfig.setInclude("tbl_user"); //设置当前参与生成的表名, 参数为可变参数
strategyConfig.setTablePrefix("tbl_"); //设置数据库表的前缀名称, 模块名 = 数据库表名 - 前缀名
例如: User = tbl_user - tbl_
strategyConfig.setRestControllerStyle(true); //设置是否启用Rest风格
strategyConfig.setVersionFieldName("version"); //设置乐观锁字段名
strategyConfig.setLogicDeleteFieldName("deleted"); //设置逻辑删除字段名
strategyConfig.setEntityLombokModel(true); //设置是否启用lombok
autoGenerator.setStrategy(strategyConfig);
```

说明：在资料中也提供了CodeGenerator代码生成器类，根据实际情况修改后可以直接使用。