

# MyBatisPlus

## 1、MybatisPlus入门案例

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.1</version>[<br/>
</dependency>
```

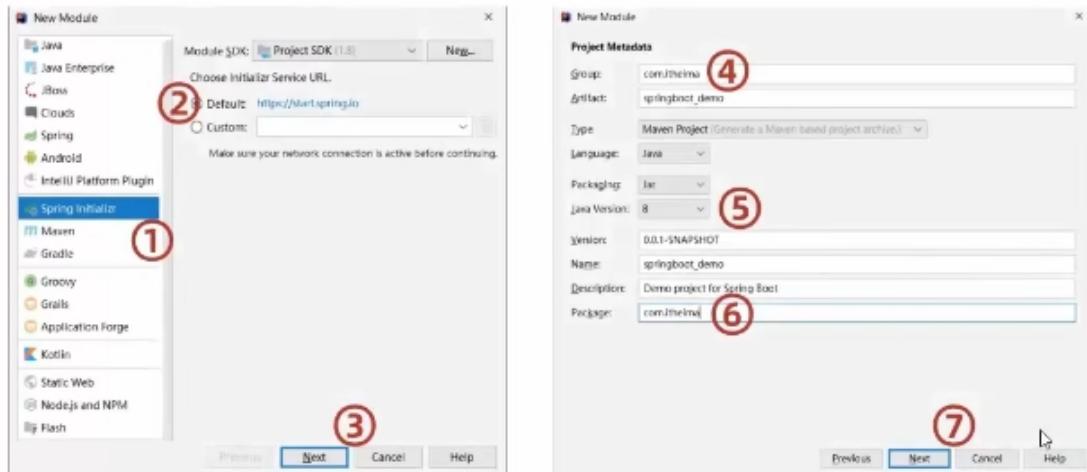
- MyBatisPlus（简称MP）是基于MyBatis框架基础上开发的增强型工具，旨在简化开发、提高效率
- 开发方式
  - 基于MyBatis使用MyBatisPlus
  - 基于Spring使用MyBatisPlus
    - ↳
    - 基于SpringBoot使用MyBatisPlus

### 入门案例

- SpringBoot整合MyBatis开发过程（复习）
  - 创建SpringBoot工程
  - 勾选配置使用的技术
  - 设置dataSource相关属性（JDBC参数）
  - 定义数据层接口映射配置

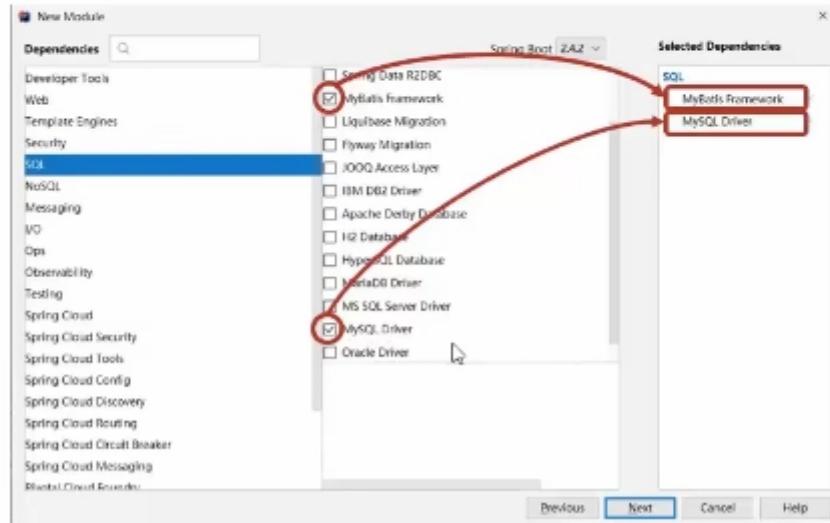
- SpringBoot整合MyBatis开发过程（复习）

- 创建SpringBoot工程
- 勾选配置使用的技术
- 设置dataSource相关属性（JDBC参数）
- 定义数据层接口映射配置



- SpringBoot整合MyBatis开发过程（复习）

- 创建SpringBoot工程
- 勾选配置使用的技术
- 设置dataSource相关属性（JDBC参数）
- 定义数据层接口映射配置



- SpringBoot整合MyBatis开发过程（复习）

- 创建SpringBoot工程
- 勾选配置使用的技术
- 设置dataSource相关属性（JDBC参数）
- 定义数据层接口映射配置

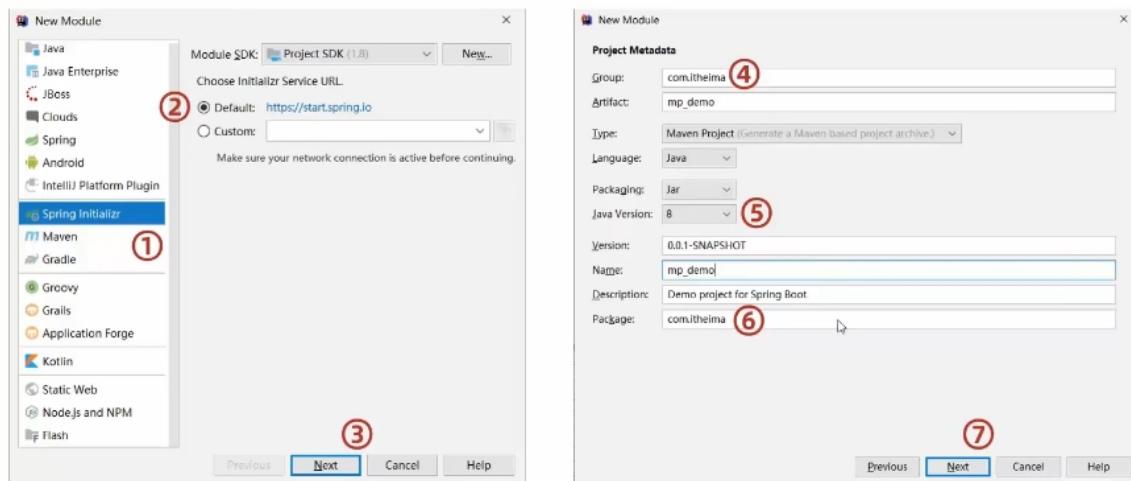
```
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mybatisplus_db?serverTimezone=UTC
    username: root
    password: root
```

- SpringBoot整合MyBatis开发过程（复习）

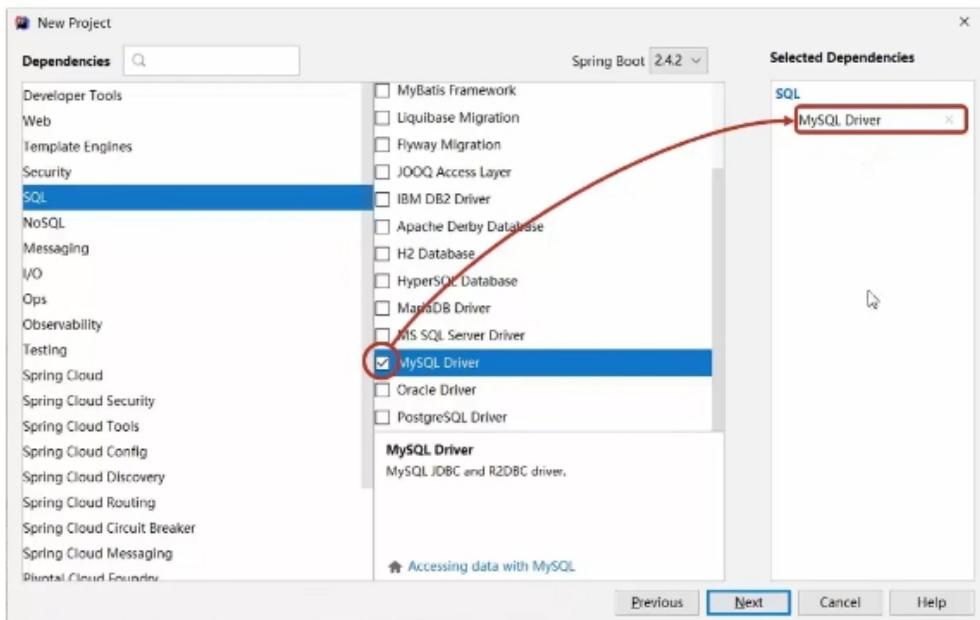
- 创建SpringBoot工程
- 勾选配置使用的技术
- 设置dataSource相关属性（JDBC参数）
- 定义数据层接口映射配置

```
@Mapper
public interface UserDao {
    @Select("select * from user where id=#{id}")
    public User getById(Long id);
}
```

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mp起步依赖

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.1</version>
</dependency>
```

### 注意事项

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

## ④：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
```

### 注意事项

如果使用Druid数据源，需要导入对应坐标

⑥：定义数据接口，继承**BaseMapper<User>**

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

⑤：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 TABLE `user` (  
    `id`        bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号' ,  
    `name`      varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '用户名' ,  
    `password`  varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '密码' ,  
    `age`       int(3) NOT NULL COMMENT '龄年' ,  
    `tel`       varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL COMMENT '电话' ,  
    PRIMARY KEY (`id`)  
)
```

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

⑥：定义数据接口，继承**BaseMapper<User>**

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

⑦：测试类中注入dao接口，测试功能

```
@SpringBootTest
class Mybatisplus01QuickstartApplicationTests {
    @Autowired
    private UserDao userDao;
    @Test
    public void testGetAll() {
        List<User> userList = userDao.selectList(null);
        userList.forEach(System.out::println);
    }
}
```



## 1. 入门程序

## 2、MyBatisPlus简介

### MyBatisPlus概述

- 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/>

## 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
- .....



小结

1. MyBatisPlus概述



1. 入门案例
2. MyBatisPlus概述

### 3、标准CRUD制作

标准数据层CRUD功能

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

↳

```
package com.itheima;

import com.baomidou.mybatisplus.core.metadata.IPage;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.itheima.dao.UserDao;
import com.itheima.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("黑马程序员");
        user.setPassword("itheima");
        user.setAge(12);
        user.setTel("4006184000");
        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 test GetById(){
        User user = userDao.selectById(2L);
        System.out.println(user);
    }

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

```

- 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>
    <scope>provided</scope>
</dependency>

```

- 常用注解: @Data

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

- 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

»

## 小结

1. 标准数据层CRUD功能

2. lombok

## 4、标准分页功能制作

### 分页功能

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

## 1、开拦截器

```
package com.itheima.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 MpConfig {
    @Bean
    public MybatisPlusInterceptor mpInterceptor(){
        //1.定义Mp拦截器
        MybatisPlusInterceptor mpInterceptor = new MybatisPlusInterceptor();
        //2.添加具体的拦截器
        mpInterceptor.addInnerInterceptor(new PaginationInnerInterceptor());
        return mpInterceptor;
    }
}
```

## 2、分页查询具体实现

```
@Test
void testGetByPage(){
    //IPage对象封装了分页操作相关的数据
    IPage page = new Page(2,3);
    userDao.selectPage(page,null);
    System.out.println("当前页码值: "+page.getCurrent());
    System.out.println("每页显示数: "+page.getSize());
    System.out.println("一共有多少页: "+page.getPages());
    System.out.println("一共有多少条数据: "+page.getTotal());
    System.out.println("数据: "+page.getRecords());//当前页的数据
}
```

## 3、启动日志

```
# 开启mp的日志（输出到控制台）
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdoutImpl
```

## 小结

1. 分页功能
2. 开启MyBatisPlus日志

## 总结

1. 标准数据层CRUD功能
2. 分页功能
3. 开启MyBatisPlus日志

## 5、条件查询的三种格式

---



# DQL编程控制

- 条件查询方式
- 查询投影
- 查询条件设定
- 字段映射与表名映射

## 条件查询

- 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

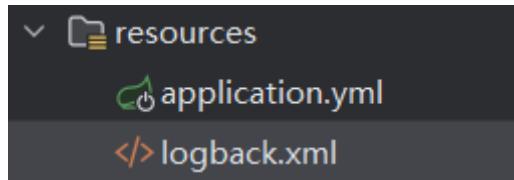
## 条件查询

- 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、如果想关闭main的启动日志

可以在



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

## 2、如果想把logo关掉

yml里写上

```
main:
  banner-mode: off
# mp日志
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdoutImpl
  global-config:
    banner: false
```

## 3、查询

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

    //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);

    //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);

//并且与或者关系
//           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);
//           //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. 条件查询
2. 条件添加方式
3. 组合查询条件

## 6、条件查询null判定

### 条件查询—null值处理

代码:

```
//模拟页面传递过来的查询数据
//UserQuery uq = new UserQuery();
//uq.setAge(10);
//uq.setAge2(30);

//null判定
//LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//lqw.lt(User::getAge, uq.getAge2());
//if( null != uq.getAge() ) {
//    lqw.gt(User::getAge, uq.getAge());
//}
//List<User> userList = userDao.selectList(lqw);
//System.out.println(userList);

//LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//先判定第一个参数是否为true, 如果为true连接当前条件
////lqw.lt(null != uq.getAge2(),User::getAge, uq.getAge2());
////lqw.gt(null != uq.getAge(),User::getAge, uq.getAge());
//lqw.lt(null != uq.getAge2(),User::getAge, uq.getAge2())
//    .gt(null != uq.getAge(),User::getAge, uq.getAge());
//List<User> userList = userDao.selectList(lqw);
//System.out.println(userList);
```

## 小结

1. 条件查询null值判定

## 7、查询投影

```
//查询投影
//LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
```

```
//      lqw.select(User::getId,User::getName,User::getAge);  
//      QueryWrapper<User> lqw = new QueryWrapper<User>();  
//      lqw.select("id","name","age","tel");  
//      List<User> userList = userDao.selectList(lqw);  
//      System.out.println(userList);  
  
//      QueryWrapper<User> lqw = new QueryWrapper<User>();  
//      lqw.select("count(*) as count, tel");  
//      //返回的是字段的个数  
//      lqw.groupBy("tel");  
//      //分组查询  
//      List<Map<String, Object>> userList = userDao.selectMaps(lqw);  
//      System.out.println(userList);
```

```
[{count=1, tel=16688886666}, {count=1, tel=18812345678}, {count=1, tel=18866668888}, {count=3, tel=4006184000}]
```

## 小结

### 1. 查询投影

## 8、查询条件设置

## 查询条件

- 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)
- .....

```
//条件查询
//      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);

//      LambdaQueryWrapper<User> lqw = new LambdaQueryWrapper<User>();
//      //范围查询 lt le gt ge eq between
//between 记得是前大于等于后小于等于 前面的是带等号右边是不带等号的
//      lqw.between(User::getAge,10,30);
//      List<User> userList = userDao.selectList(lqw);
//      System.out.println(userList);

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

## 查询条件

- 更多查询条件设置参看<https://mybatis-plus.gitee.io/zh/doc/api/wrapper.html#abstractwrapper>

## 1. 查询条件

- 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)
- .....

## 2. 查询API

# 9、映射匹配兼容性

## 字段映射与表名映射

- 问题一：表字段与编码属性设计不同步

```
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;
    private String password; 我改
    private Integer age;
    private String tel;
}
```



## 字段映射与表名映射

```
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;
}
```

## 字段映射与表名映射

- 名称: @TableField
- 类型: 属性注解
- 位置: 模型类属性定义上方
- 作用: 设置当前属性对应的数据库表中的字段关系
- 范例:

```
public class User {
    @TableField(value="pwd")
    private String password;
}
```

- 相关属性
  - ◆ value (默认) : 设置数据库表字段名称

- 问题二: 编码中添加了数据库中未定义的属性

```
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;
    private Integer online;
}
```

- 问题二：编码中添加了数据库中未定义的属性

```
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;
}
```

## 字段映射与表名映射

- 名称：@TableField
- 类型：**属性注解**
- 位置：模型类属性定义上方
- 作用：设置当前属性对应的数据表中的字段关系
- 范例：

```
public class User {
    @TableField(exist = false)
    private Integer online;
}
```

- 相关属性
  - ◆ value：设置数据库表字段名称
  - ◆ exist：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

## 字段映射与表名映射

- 问题三：采用默认查询开放了更多的字段查看权限

```
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")
    private String password;
    private Integer age;
    private String tel;
    @TableField(exist = false)
    private Integer online;
}
```

### 字段映射与表名映射

- 名称: @TableField
- 类型: 属性注解
- 位置: 模型类属性定义上方
- 作用: 设置当前属性对应的数据库表中的字段关系
- 范例:

```
public class User {
    @TableField(value="pwd",select = false)
    private String password;
}
```

- 相关属性
  - ◆ value: 设置数据库表字段名称
  - ◆ exist: 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
  - ◆ select: 设置属性是否参与查询, 此属性与select()映射配置不冲突

### 字段映射与表名映射

- 问题四: 表名与编码开发设计不同步

```
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`)
)
```

```
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;
}
```

### 字段映射与表名映射

- 问题四: 表名与编码开发设计不同步

```
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;
}
```

## 字段映射与表名映射

- 名称: @TableName
- 类型: **类注解**
- 位置: 模型类定义上方
- 作用: 设置当前类对应与数据库表关系
- 范例:

```
@TableName("tbl_user")
public class User {
    private Long id;
}
```

- 相关属性
  - ◆ value: 设置数据库表名称



# 小结

1. 字段映射与表名映射



# 总结

1. 条件查询
2. 查询映射
3. 查询条件
4. 字段映射与表名映射

## 10、id生成策略

---



### DML编程控制

- Insert
- Delete
- Update

### id生成策略控制

- 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
  - ◆ .....

## id生成策略控制

- 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
- 范例:

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

- 相关属性
  - ◆ value: 设置数据库主键名称
  - ◆ type: 设置主键属性的生成策略, 值参照IdType枚举值

## id生成策略控制

- 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

0 | 00100110111011010101100001101010011000110 | 10000 | 10001 | 000000000010  
占位符: 0                           时间戳(41)                           机器码(5+5)                   序列号(12)

## 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;  
}
```

## 全局配置

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



1. id生成策略控制
2. 全局设置

## 11、多数据操作（删除与查询）

## 多记录操作



```
@Test
void testDelete(){
    //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);
    //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);

    userDao.deleteById(2L);
    System.out.println(userDao.selectList(null));
}
```

# 小结

## 1. 多记录操作

## 12、逻辑删除

### 逻辑删除

- 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
1	2025/4/1	100,000.00	1
2	2025/5/12	300,000.00	1
3	2025/9/11	500,000.00	1
4	2025/11/14	80,000.00	2
5	2025/12/25	20,000.00	3

员工编号	姓名	工号	deleted
1	张业绩	9501	1
2	李小白	9502	0
3	王笑笑	9503	0

编号	姓名	业绩
1	李小白	80,000.00
2	王笑笑	20,000.00
合计		100,000.00

## 逻辑删除

- 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
1	2025/4/1	100,000.00	1
2	2025/5/12	300,000.00	1
3	2025/9/11	500,000.00	1
4	2025/11/14	80,000.00	2
5	2025/12/25	20,000.00	3

员工编号	姓名	工号	deleted
1	张业绩	9501	1
2	李小白	9502	0
3	王笑笑	9503	0

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

### 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						
id						
name						
pwd						
age						
tel						
* deleted						



### 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

```
public class User {  
    private Long id;  
    @TableLogic  
    private Integer deleted;  
}
```

③：配置逻辑删除字面值

```
mybatis-plus:  
  global-config:  
    db-config:  
      logic-delete-field: deleted  
      logic-not-delete-value: 0  
      logic-delete-value: 1
```

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



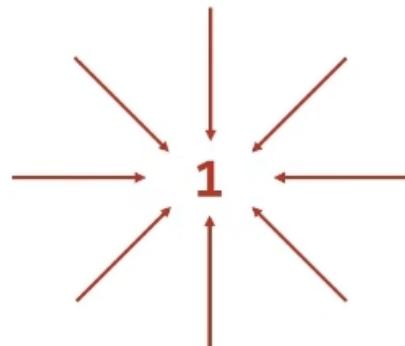
## 1. 逻辑删除

# 13、乐观锁

int的最大范围是11位数字

## 乐观锁

- 业务并发现象带来的问题：秒杀



### 步骤 乐观锁案例

- 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							
id							PK 1
name							
pwd							
age							
tel							
deleted							
version							

- 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

```
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

```
@Configuration
public class MpConfig {
    @Bean
    public MybatisPlusInterceptor mpInterceptor() {
        MybatisPlusInterceptor mpInterceptor = new MybatisPlusInterceptor();
        mpInterceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());
        return mpInterceptor;
    }
}
```

#### 4、数据修改

```
@Test
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);                //verion=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=?
```

## 小结

1. 乐观锁

## 总结

1. id自增策略控制 (Insert)
2. 多记录操作 (Delete、Select)
3. 逻辑删除 (Delete/Update)
4. 乐观锁 (Update)

## 14、代码生成器

## 代码生成器

造句：\_\_\_\_\_这妞\_\_\_\_\_正\_\_\_\_\_点\_\_\_\_\_呢。

```
package com.itheima.dao;

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

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

参数

模板

## 代码生成器

```
package com.itheima.domain;

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

@Data
@TableName("_____")
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long __;
    private String __;
    @TableField(value = "pwd",select = false)
    private String __;
    private Integer __;
    private String __;
    @TableField(exist = false)
    private Integer online;
    @TableLogic(value = "0",delval = "1")
    private Integer __;
    @Version
    private Integer __;
}
```

数据库读取  
开发者配置

## 代码生成器

```
package com.itheima.domain;

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 __;
}
```

模板

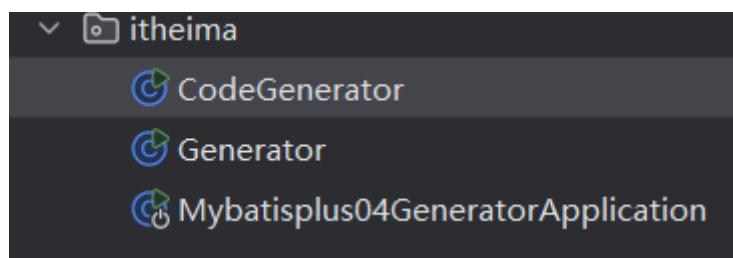
## 代码生成器

- 模板：MyBatisPlus提供
- 数据库相关配置：读取数据库获取信息
- 开发者自定义配置：手工配置

先导依赖

```
<!--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>
```



```
package com.itheima;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
import com.baomidou.mybatisplus.generator.config.GlobalConfig;
import com.baomidou.mybatisplus.generator.config.PackageConfig;
import com.baomidou.mybatisplus.generator.config.StrategyConfig;

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

```

//1.获取代码生成器的对象
AutoGenerator autoGenerator = new AutoGenerator();

//设置数据库相关配置
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);

//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("黑马程序员"); //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
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
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);
//2.执行生成操作
autoGenerator.execute();
}

}

```

然后就会生成代码



# 总结

1. 代码生成器