# MyBatis-Plus

# 学习目标

- [ ] 了解mybatisplus的特点
- [ ] 能够掌握mybatisplus快速入门
- [ ] 能够掌握mybatisplus常用注解
- [ ] 能够掌握mybatisplus常用的增删改查
- [ ] 能够掌握mybatisplus自动代码生成

# 第一章-MyBatis-Plus入门

## 知识点-MyBatis-Plus介绍

### 1.目标

- [ ] 知道什么是mybatisplus

### 2.讲解

MyBatis-Plus（简称 MP）是一个 MyBatis 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

官网：https://mybatis.plus/

| 润物无声 | 效率至上 | 丰富功能 |
| --- | --- | --- |
| 只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑。 | 只需简单配置，即可快速进行单表 CRUD 操作，从而节省大量时间。 | 代码生成、物理分页、性能分析等功能一应俱全。 |

### 3.小结

MyBatis-Plus（简称 MP）是一个 MyBatis 的增强工具, 让我们使用MyBatis开发更加的快速,方便了.

## 案例-MyBatis-Plus快速入门

### 1.需求

- [ ] 查询id为1的用户,把结果输出控制台

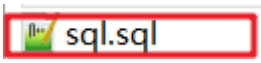
### 2.分析

1. 准备数据库
2. 创建工程, 在pom文件添加坐标(mybatisplus, 驱动等)
3. 创建配置文件(配置连接数据库的基本项, MyBatisPlus的)
4. 创建启动类, 开启mapper扫描

5. 创建pojo(需要和数据库的表进行关联)
6. 创建mapper接口继承BaseMapper
7. 测试

## 3.实现

- 准备数据库




- 创建工程, 在pom文件添加坐标

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.itheima</groupId>
    <artifactId>mp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <artifactId>spring-boot-starter-parent</artifactId>
        <groupId>org.springframework.boot</groupId>
        <version>2.1.0.RELEASE</version>
    </parent>


    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!--spring-boot起步依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <!-- mysql  驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!-- lombok  ,自动生成get,Set 方法-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <!--测试起步依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

```
        </dependency>
        <!--mybatisplus起步依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.1.1</version>
        </dependency>
    </dependencies>


</project>
```

- 创建配置文件

```
# datasource
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
    username: root
    password: 123456
    driver-class-name: com.mysql.jdbc.Driver
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl #打印sql语句
```

- 创建启动类, 开启mapper扫描

```
package com.itheima.mp;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * @Description:
 * @author: yp
 */
@SpringBootApplication
@MapperScan("com.itheima.mp.mapper")
public class MpApllication {

    public static void main(String[] args) {
        SpringApplication.run(MpApllication.class,args);
    }

}
```

- 创建pojo

```
package com.itheima.mp.pojo;

import com.baomidou.mybatisplus.annotation.TableName;
```

```
import lombok.Data;

import java.io.Serializable;

@TableName("tb_user")
@Data
public class User implements Serializable {
    private Long id;
    private String username;
    private String password;
    private String name;
    private Integer age;
    private String email;
}
```

- 创建UserMapper继承BaseMapper

```
package com.itheima.mp.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.itheima.mp.pojo.User;

/**
 * @Description:
 * @author: yp
 */
public interface UserMapper extends BaseMapper<User> {
}
```

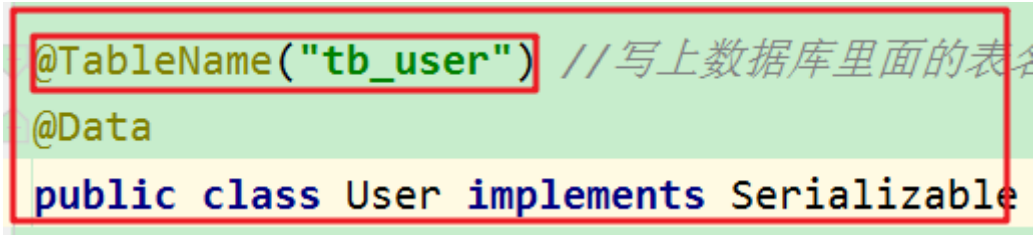
- 测试

```
package com.itheima.test;

import com.itheima.mp.MpApllication;
import com.itheima.mp.mapper.UserMapper;
import com.itheima.mp.pojo.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

/**
 * @Description:
 * @author: yp
 */
@SpringBootTest(classes = MpApllication.class)
@RunWith(SpringRunner.class)
public class MpTest {

    @Autowired
    private UserMapper userMapper;
```

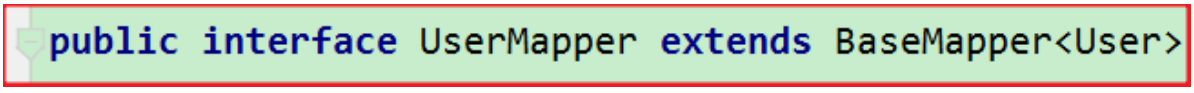
```
    @Test
    public void fun01(){
        User user = userMapper.selectById(1);
        System.out.println(user);
    }

}
```

## 4.小结

1. 在pojo里面需要和表进行映射




2. Mapper接口继承BaseMapper接口




# 第二章-Mybatis-Plus的练习【重点】

## 知识点-新增

### 1.目标

- [ ] 使用Mybatis-Plus向tb_user新增一条记录

### 2.路径

1. 新增练习
2. 实体类常用注解
3. 常用配置

### 3.讲解

#### 3.1. 新增

```java
// 插入一条记录
int insert(T entity);
```

```java
@Test
//新增
public void fun02(){
    User user = new User();
    user.setId(111L);
    user.setUsername("wbb");
    user.setPassword("123456");
    user.setName("王八");
    user.setAge(18);
    user.setEmail("wb@163.com");
    int rows = userMapper.insert(user);
    System.out.println(rows);
}
```

### 3.2. 实体类常用注解

https://mp.baomidou.com/guide/annotation.html#sqlparser

- @TableName（）： 表名注解 实体类和表进行映射

| 属性 | 类型 | 必须指定 | 默认值 |
| --- | --- | --- | --- |
| value | String | 否 | "" |

- @TableId()：主键注解

| 属性 | 类型 | 必须指定 | 默认值 | 描述 |
| --- | --- | --- | --- | --- |
| value | String | 否 | "" | 主键字段名 |
| type | Enum | 否 | IdType.NONE | 主键类型 |

- id的type

| 值 | 描述 |
| --- | --- |
| AUTO | 数据库ID自增 |
| NONE | 无状态,该类型为未设置主键类型(注解里等于跟随全局,全局里约等于 INPUT) |
| INPUT | insert前自行set主键值 |
| ASSIGN_ID | 分配ID(主键类型为Number(Long和Integer)或String)(since 3.3.0),使用接口`IdentifierGenerator`的方法`nextId`(默认实现类为`DefaultIdentifierGenerator`雪花算法) |
| ASSIGN_UUID | 分配UUID,主键类型为String(since 3.3.0),使用接口`IdentifierGenerator`的方法`nextUUID`(默认default方法) |
| ID_WORKER | 分布式全局唯一ID 长整型类型(please use `ASSIGN_ID`) |
| UUID | 32位UUID字符串(please use `ASSIGN_UUID`) |
| ID_WORKER_STR | 分布式全局唯一ID 字符串类型(please use `ASSIGN_ID` |

- @TableField()：字段注解(非主键)

| 属性 | 类型 | 必须指定 | 默认值 | 描述 |
| --- | --- | --- | --- | --- |
| value | String | 否 | "" | 数据库字段名 |
| exist | boolean | 否 | true | 是否为数据库表字段 |
| fill | Enum | 否 | FieldFill.DEFAULT | 字段自动填充策略 |

实体类的属性名和数据库的字段名自动映射：

1. 名称一样
2. 数据库字段使用_分割，实体类属性名使用驼峰名称

否则需要使用 @TableField(value="列名") 指定映射关系

### 3.3 常用配置

```
mybatis-plus:
  global-config:
    db-config:
      # 表名前缀
      table-prefix: tb_
      # id生成策略 数据库自增
      id-type: auto
  configuration:
    # 日志
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

# 知识点-更新和删除

## 1.目标

- [ ] 能够使用MyBatis-Plus完成更新和删除

## 2.路径

1. 更新
    - 根据id更新
    - 根据条件更新
2. 删除
    - 根据id删除
    - 根据id批量删除
    - 根据条件删除

# 3.讲解

## 3.1更新

```java
// 根据 whereEntity 条件，更新记录
int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T> updateWrapper);
// 根据 ID 修改
int updateById(@Param(Constants.ENTITY) T entity);
```

- 根据id更新

```
    @Test
    //根据id更新   把id为18的用户名字改成小红红【一般是先查询再更新】
    public void fun03(){
        User user = new User();
        user.setId(18L);
        user.setUsername("小红红");
        userMapper.updateById(user);
    }
```

- 根据条件更新

```
    @Test
    //根据条件更新    把名字是小红红的用户的密码改成88888
    public void fun04(){

        //userMapper.update(更新后的数据,构造的条件);
        //1.更新后的数据
        User user = new User();
        user.setPassword("88888");

        //2.构造的条件
        UpdateWrapper<User> updateWrapper = new UpdateWrapper<User>();
        updateWrapper.eq("username","小红红");

        userMapper.update(user,updateWrapper);
    }
```

## 3.2删除

```java
// 根据 entity 条件，删除记录
int delete(@Param(Constants.WRAPPER) Wrapper<T> wrapper);
// 删除（根据ID 批量删除）
int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
// 根据 ID 删除
int deleteById(Serializable id);
// 根据 columnMap 条件，删除记录
int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
```

- 根据id删除

```
@Test
 //根据id删除   删除id为18的用户
 public void fun05(){
     userMapper.deleteById(18);
 }
```

- 根据id集合批量删除

```
 //根据id批量删除   删除id为5,6,7的
 @Test
 public void fun06(){
     List list = new ArrayList();
     list.add(5L);
     list.add(6L);
     list.add(7L);
     userMapper.deleteBatchIds(list);
 }
```

- 根据条件删除

```
 //根据条件删除 删除name=赵六
 @Test
 public void fun07(){
     UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();
     //updateWrapper.eq("name","赵六").or().ge("age",18);
     updateWrapper.eq("name","赵六");
     userMapper.delete(updateWrapper);

 }
```

# 知识点-查询

## 1.目标

- [ ] 能够使用MyBatis-Plus完成查询

## 2.路径

1. 基础条件查询
2. 模糊查询
3. 逻辑查询
4. 排序
5. select指定查询的字段
6. 分页查询

## 3.讲解

```java
// 根据 ID 查询
T selectById(Serializable id);
// 根据 entity 条件，查询一条记录
T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);

// 查询（根据ID 批量查询）
List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
// 根据 entity 条件，查询全部记录
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 查询（根据 columnMap 条件）
List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
// 根据 Wrapper 条件，查询全部记录
List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录。注意： 只返回第一个字段的值
List<Object> selectObjs(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);

// 根据 entity 条件，查询全部记录（并翻页）
IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录（并翻页）
IPage<Map<String, Object>> selectMapsPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> que
// 根据 Wrapper 条件，查询总记录数
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

## 3.1 基础条件查询

通过 QueryWrapper 指定查询条件

```
eq( ) :  等于 =
ne( ) :  不等于 <>
gt( ) :  大于 >
ge( ) :  大于等于  >=
lt( ) :  小于 <
le( ) :  小于等于 <=
between ( ) :  BETWEEN 值1 AND 值2
notBetween ( ) :  NOT BETWEEN 值1 AND 值2
in( ) :  in
notIn( ) : not in
```

```java
    //基本条件查询  查询name=张三的用户信息
    @Test
    public void fun08(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("name","张三");
        //queryWrapper.gt("age",16); //查询age>16

        //2.调用方法
        //userMapper.selectOne(queryWrapper); //查询1个的情况
        List<User> list = userMapper.selectList(queryWrapper);//查询多个的情况
        if(list.size()==1){
            System.out.println(list.get(0));
        }else{
            System.out.println(list);
        }
    }
```

### 3.2 模糊查询 like

```
like   %% 【索引失效】  数量大时
notLike <> 【索引失效】
likeLeft %s  LIKE '%值' 【坚决不写,让索引失效】
likeRight s%  LIKE '值%' 【可写】
```

```
    //模糊查询  查询姓张的用户
    @Test
    public void fun09(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.likeRight("name","张"); // 相当于 like 张%
        //queryWrapper.like("name","张"); // 相当于 like %张%  查询名字里面包含张的
        //2.调用方法
        List<User> list = userMapper.selectList(queryWrapper);
        System.out.println(list);
    }
```

### 3.3逻辑查询 or

```
or( ) : 让紧接着下一个方法用or连接
```

```
    //逻辑查询  多条件 and  查询姓张的 并且age>15岁的
    @Test
    public void fun10(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.likeRight("name","张").gt("age",15);

        //2.调用方法
        List<User> list = userMapper.selectList(queryWrapper);
        System.out.println(list);
    }

    //逻辑查询   or  查询姓张的或者age<22岁的
    @Test
    public void fun11(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.likeRight("name","张").or().lt("age",22);

        //2.调用方法
        List<User> list = userMapper.selectList(queryWrapper);
        System.out.println(list);
    }
```

### 3.4. 排序查询

```
orderBy
orderByAsc   升序
orderByDesc  降序
```

```
    //排序查询    or  查询姓张的或者age<22岁的. 根据age降序查询, 如果age一样根据id降序
    @Test
    public void fun12(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();

 queryWrapper.likeRight("name","张").or().lt("age",22).orderByDesc("age").orderBy
Desc("id");
        //2.调用方法
        List<User> list = userMapper.selectList(queryWrapper);
        System.out.println(list);
    }
```

### 3.5. select指定需要查询的字段

```
    //查询username和password两个列
    @Test
    public void fun13(){
        //1.构造查询条件
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.select("username","password");
        //2.调用方法
        List<User> list = userMapper.selectList(queryWrapper);
        System.out.println(list);
    }
```

### 3.6. 分页条件查询

limit a,b;

a=从哪里开始

b=一页查询多少条

传什么过来? 传查询哪一页, 一页查询多少条

#### Page

```java
// 无条件分页查询
IPage<T> page(IPage<T> page);
// 条件分页查询
IPage<T> page(IPage<T> page, Wrapper<T> queryWrapper);
// 无条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page);
// 条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page, Wrapper<T> queryWrapper);
```

1. 注册分页拦截器

```
package com.itheima.mp.config;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class PageConfig {

    /**
     * 3.4.0之前的版本用这个
     * @return
     */
    @Bean
    public PaginationInterceptor paginationInterceptor(){
        return  new PaginationInterceptor();
    }

    /**
     * 3.4.0之后提供的拦截器的配置方式
     * @return
     */
  /* @Bean
   public MybatisPlusInterceptor mybatisPlusInterceptor(){
       MybatisPlusInterceptor mybatisPlusInterceptor = new
MybatisPlusInterceptor();
       mybatisPlusInterceptor.addInnerInterceptor(new
PaginationInnerInterceptor());
       return mybatisPlusInterceptor;
   }*/
}
```

2. 代码

```
    //分页条件查询
    @Test
    public void fun14(){
        //1.构造分页对象Page(封装查询哪一页和一页查询多少 eg: 参数是1,2 查询第一页, 查询2
条)
        Page<User> page = new Page<>(1, 2);

        //2.构造条件对象
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.gt("age",15);

        //3.调用selectPage()方法  返回值就是分页查询结果的对象(包含了总数量,查询的结果
List...)
        IPage<User> pageResult = userMapper.selectPage(page, queryWrapper);
        System.out.println("总数量="+pageResult.getTotal());
        System.out.println("查询的数据="+pageResult.getRecords());

    }
```

## 4.小结

1. API不要记, 猜. 在IDEA里面 mapper对象. 方法名全部出来了

# 知识点-LambdaWrapper

## 1.目标

- [ ] 掌握LambdaWrapper的使用

## 2.路径

1. LambdaQueryWrapper
2. LambdaUpdateWrapper

## 3.讲解

- LambdaQueryWrapper

```
   //LambdaQueryWrapper  别人敲了的项目
    @Test
    public void fun15(){
        //1.构造分页对象Page(封装查询哪一页和一页查询多少 eg: 参数是1,2 查询第一页, 查询2
条)
        Page<User> page = new Page<>(1, 2);

        //2.构造条件对象
        //QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        //queryWrapper.gt("age",15);

        //方式一: LambdaQueryWrapper
        //LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
        //queryWrapper.gt(User::getAge,15);
        //IPage<User> pageResult = userMapper.selectPage(page, queryWrapper);

        //3.调用selectPage()方法  返回值就是分页查询结果的对象(包含了总数量,查询的结果
List...)
        //方式二: 使用静态方法
        IPage<User> pageResult = userMapper.selectPage(page, Wrappers.
<User>lambdaQuery().gt(User::getAge,15));

        System.out.println("总数量="+pageResult.getTotal());
        System.out.println("查询的数据="+pageResult.getRecords());

    }
```

- LambdaUpdateWrapper

```
    @Test
    //根据条件更新
    public void fun15(){
        //设置更新后的数据
        User user = new User();
        user.setPassword("777777");
```

```
        //构造条件
        //UpdateWrapper<User> wrapper = new UpdateWrapper<>();
        //wrapper.eq("user_name","tq");

        LambdaUpdateWrapper<User> wrapper = new LambdaUpdateWrapper<>();
        wrapper.eq(User::getUserName,"tq");
        int rows = userMapper.update(user, wrapper);
        System.out.println(rows);
    }
```

### 4.小结

1. 能够看的懂, 如果写不习惯, 不要求. 练一遍

# 第三章-Service 封装

## 知识点-Service 封装

### 1.目标

Mybatis-Plus 为了开发更加快捷，对业务层也进行了封装，直接提供了相关的接口和实现类。我们在进行业务层开发时，可以继承它提供的接口和实现类，使得编码更加高效

- [ ] 掌握Service 封装

### 2.步骤

1. 定义接口继承IService
2. 定义实现类继承ServiceImpl<Mapper，Entity> 实现定义的接口

> 提供的仅仅是基本的CRUD业务, 复杂的业务一样需要自己定义方法,自己实现

### 3.实现

- 接口

```
package com.itheima.mp.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.itheima.mp.pojo.User;

/**
 * @Description:
 * @author: yp
 */
public interface UserService extends IService<User> {
}
```

- 实现类

```java
package com.itheima.mp.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.itheima.mp.mapper.UserMapper;
import com.itheima.mp.pojo.User;
import com.itheima.mp.service.UserService;
import org.springframework.stereotype.Service;

/**
 * @Description:
 * @author: yp
 */
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements
UserService {

}
```

- 测试

```java
    @Autowired
    private UserService userService;
    @Test
    public void fun16(){
        System.out.println(userService.getById(1L));
    }
```

# 第四章-逆向工程-代码生成器

## 知识点-AutoGenerator

### 1.目标

- [ ] 了解AutoGenerator使用

### 2.路径

1. AutoGenerator的介绍
2. AutoGenerator的使用

### 3.讲解

#### 3.1AutoGenerator的介绍

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Entity、Mapper、Mapper XML、Service、Controller 等各个模块的代码，极大的提升了开发效率。

## 3.2AutoGenerator的使用

### 3.2.1步骤

1. 创建工程导入坐标
2. 执行main方法

### 3.2.2使用

- 创建工程导入坐标

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.itheima</groupId>
    <artifactId>my-springboot-autogenerator</artifactId>
    <version>1.0-SNAPSHOT</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
            <version>5.1.47</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--mybatis plus 起步依赖-->

        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.1.1</version>
        </dependency>

        <!--mp 代码生成器-->
```

```
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.1.1</version>
        </dependency>

        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
            <version>2.3.30</version>
        </dependency>
    </dependencies>
</project>
```

- 执行main方法

```
package com.itheima;

import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;
import com.baomidou.mybatisplus.core.toolkit.StringPool;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.InjectionConfig;
import com.baomidou.mybatisplus.generator.config.*;
import com.baomidou.mybatisplus.generator.config.po.TableInfo;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

// 演示例子，执行 main 方法控制台输入模块表名回车自动生成对应项目目录中
public class CodeGenerator {

    /**
     * <p>
     * 读取控制台内容
     * </p>
     */
    public static String scanner(String tip) {
        Scanner scanner = new Scanner(System.in);
        StringBuilder help = new StringBuilder();
        help.append("请输入" + tip + "：");
        System.out.println(help.toString());
        if (scanner.hasNext()) {
            String ipt = scanner.next();
            if (StringUtils.isNotEmpty(ipt)) {
                return ipt;
            }
        }
        throw new MybatisPlusException("请输入正确的" + tip + "！");
    }

    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();
```

```
        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        System.out.println(projectPath);
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("lh");
        gc.setOpen(false);
        // gc.setSwagger2(true); 实体属性 Swagger2 注解
        mpg.setGlobalConfig(gc);

        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/mydb?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC");
        // dsc.setSchemaName("public");
        dsc.setDriverName("com.mysql.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        mpg.setDataSource(dsc);

        // 包配置
        PackageConfig pc = new PackageConfig();
        pc.setModuleName(scanner("模块名"));
        pc.setParent("com.itheima");
        mpg.setPackageInfo(pc);

        // 自定义配置
        InjectionConfig cfg = new InjectionConfig() {
            @Override
            public void initMap() {
                // to do nothing
            }
        };

        // 如果模板引擎是 freemarker
        String templatePath = "/templates/mapper.xml.ftl";
        // 如果模板引擎是 velocity
        // String templatePath = "/templates/mapper.xml.vm";

        // 自定义输出配置
        List<FileOutConfig> focList = new ArrayList<>();
        // 自定义配置会被优先输出
        focList.add(new FileOutConfig(templatePath) {
            @Override
            public String outputFile(TableInfo tableInfo) {
                // 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟
着发生变化！！
                return projectPath + "/src/main/resources/mapper/" +
pc.getModuleName()
                        + "/" + tableInfo.getEntityName() + "Mapper" +
StringPool.DOT_XML;
            }
        });
        /*
        cfg.setFileCreate(new IFileCreate() {
            @Override
```

```
            public boolean isCreate(ConfigBuilder configBuilder, FileType
fileType, String filePath) {
                // 判断自定义文件夹是否需要创建
                checkDir("调用默认方法创建的目录，自定义目录用");
                if (fileType == FileType.MAPPER) {
                    // 已经生成 mapper 文件判断存在，不想重新生成返回 false
                    return !new File(filePath).exists();
                }
                // 允许生成模板文件
                return true;
            }
        });
        */
        cfg.setFileOutConfigList(focList);
        mpg.setCfg(cfg);

        // 配置模板
        TemplateConfig templateConfig = new TemplateConfig();

        // 配置自定义输出模板
        //指定自定义模板路径，注意不要带上.ftl/.vm, 会根据使用的模板引擎自动识别
        // templateConfig.setEntity("templates/entity2.java");
        // templateConfig.setService();
        // templateConfig.setController();

        templateConfig.setXml(null);
        mpg.setTemplate(templateConfig);

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        //strategy.setSuperEntityClass("你自己的父类实体,没有就不用设置!");
        strategy.setEntityLombokModel(true);
        strategy.setRestControllerStyle(true);
        // 公共父类
        //strategy.setSuperControllerClass("你自己的父类控制器,没有就不用设置!");
        // 写于父类中的公共字段
        strategy.setSuperEntityColumns("id");
        strategy.setInclude(scanner("表名，多个英文逗号分割").split(","));
        strategy.setControllerMappingHyphenStyle(true);
        strategy.setTablePrefix(pc.getModuleName() + "_");
        strategy.setTablePrefix("tb_"); //去掉表名前缀
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }
}
```