# MyBatisPlus

## 简介

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus" \t "_blank)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/" \t "_blank) 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。

我们的愿景是成为 MyBatis 最好的搭档，就像 [魂斗罗](https://mp.baomidou.com/img/contra.jpg) 中的 1P、2P，基友搭配，效率翻倍。



## 代码及文档发布地址

官方地址:

<http://mp.baomidou.com>

代码发布地址:

Github: <https://github.com/baomidou/mybatis-plus>

Gitee: <https://gitee.com/baomidou/mybatis-plus>

文档发布地址:

[http://mp.baomidou.com/#/?id=%E7%AE%80%E4%BB%8B](http://mp.baomidou.com/" \l "/?id=%E7%AE%80%E4%BB%8B)

## 快速入门

### 环境准备

Eclipse

Maven3.5+

SpringBoot 2.1.5

JDK1.8+

### 数据库

|  |
| --- |
| DROP TABLE IF EXISTS USER;  CREATE TABLE USER  (  id BIGINT(20) NOT NULL AUTO\_INCREMENT COMMENT '主键ID',  NAME VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',  age INT(11) NULL DEFAULT NULL COMMENT '年龄',  email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',  PRIMARY KEY (id)  );  DELETE FROM USER;  INSERT INTO USER (id, NAME, age, email) VALUES  (1, 'Jone', 18, 'test1@baomidou.com'),  (2, 'Jack', 20, 'test2@baomidou.com'),  (3, 'Tom', 28, 'test3@baomidou.com'),  (4, 'Sandy', 21, 'test4@baomidou.com'),  (5, 'Billie', 24, 'test5@baomidou.com'); |

### 创建springboot工程

#### 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!-- MySQL驱动包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <!-- MyBatisPlus驱动包 -->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.1.1</version>  </dependency>  <!-- 简化实体类，可选 -->  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>  <!-- druid 数据库连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency> |

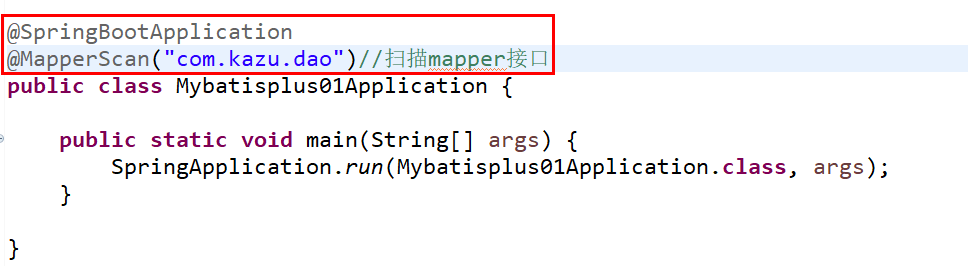
#### application.properties配置文件

|  |
| --- |
| #加载驱动  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  #数据库连接路径  spring.datasource.url=jdbc:mysql://localhost:3306/db\_mybatisplus?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  #数据库用户名  spring.datasource.username=root  #数据库密码  spring.datasource.password=root  #数据源类型（阿里巴巴）  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource |

#### 实体类

|  |
| --- |
| **public** **class** User {  **private** Long id;  **private** String name;  **private** Integer age;  **private** String email;  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  **public** String getEmail() {  **return** email;  }  **public** **void** setEmail(String email) {  **this**.email = email;  }  @Override  **public** String toString() {  **return** "User [id=" + id + ", name=" + name + ", age=" + age + ", email=" + email + "]";  }  } |

#### 启动类



#### Mapper接口

|  |
| --- |
| /\*\*  \* 自定义的Mapper接口通过继承BaseMapper，即可使用MyBatisPlus提供的通用增删改查方法  \* **@author** KazuGin  \*/ |

#### 测试类

|  |
| --- |
|  |

# MyBatisPlus的使用

## 通用CRUD的使用

1. 提出问题:

假设我们已存在一张 tbl\_employee 表， 且已有对应的实体类 Employee ， 实现tbl\_employee 表的 CRUD 操作我们需要做什么呢？

1. 实现方式:

基于Mybatis

需要编写EmployeeMapper 接口，并手动编写CRUD 方法

提供EmployeeMapper.xml 映射文件，并手动编写每个方法对应的SQL 语句.

基于MP

只需要创建 EmployeeMapper 接口, 并继承 BaseMapper 接口.这就是使用 MP

需要完成的所有操作，甚至不需要创建SQL 映射文件。

★注：数据库文件参考附件employee.sql文件

## 新增操作

### MP源码

|  |
| --- |
|  |

### 实体类

|  |
| --- |
| **public** **class** Employee {  **private** Integer id;//编号  **private** String userName;//用户名  **private** String email;//邮箱  **private** Integer gender;//性别  **private** Integer age;//年龄  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getUserName() {  **return** userName;  }  **public** **void** setUserName(String userName) {  **this**.userName = userName;  }  **public** String getEmail() {  **return** email;  }  **public** **void** setEmail(String email) {  **this**.email = email;  }  **public** Integer getGender() {  **return** gender;  }  **public** **void** setGender(Integer gender) {  **this**.gender = gender;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  } |

### Mapper接口

|  |
| --- |
| **public** **interface** EmployeeMapper **extends** BaseMapper<Employee>{  } |

### 测试类

|  |
| --- |
|  |

### 常见错误

#### 错误1：没有设置主键生成策略

|  |
| --- |
| 在实体类中添加@TableId注解即可 |

#### 错误2：实体类名称与表名称不一致

|  |
| --- |
| 在实体类加入@TableName注解  s |

#### 错误3：属性名与列名不一致

|  |
| --- |
|  |

## 常用注解

### @TableId

|  |
| --- |
|  |

### @TableName

|  |
| --- |
|  |

### @TableFiled

|  |
| --- |
|  |

## 删除操作

### MP源码

|  |
| --- |
|  |

### 测试类

#### deleteById()

|  |
| --- |
|  |

#### deleteByMap()

|  |
| --- |
|  |

#### delete()

|  |
| --- |
|  |

#### deleteBatchIds()

|  |
| --- |
|  |

## 修改操作

### MP源码

|  |
| --- |
|  |

### 测试类

#### updateById()

|  |
| --- |
|  |

#### update()

|  |
| --- |
|  |

## 查询操作

### MP源码

|  |
| --- |
|  |

### selectById()

|  |
| --- |
|  |

### selectOne()

|  |
| --- |
|  |

### selectList()

|  |
| --- |
|  |

### selectCount ()

|  |
| --- |
|  |

### selectBatchIds ()

|  |
| --- |
|  |

### selectByMap()

|  |
| --- |
|  |

### MP启动注入SQL原理

#### 问题

|  |
| --- |
| xxxMapper 继承了BaseMapper<T>, BaseMapper 中提供了通用的 CRUD 方法, 方法来源于 BaseMapper, 有方法就必须有SQL, 因为 MyBatis 最终还是需要通过SQL 语句操作数据. |

#### 分析本质

|  |
| --- |
| SqlSession  SqlSessionFactory  Configuration  mapperStatement  sqlSource |

## 条件构造器

### 简介

|  |
| --- |
| * Mybatis-Plus 通过 QueryWrapper（简称 QW，MP 封装的一个查询条件构造器）来让用户自由的构建查询条件，简单便捷，没有额外的负担， 能够有效提高开发效率 * 实体包装器，主要用于处理 sql 拼接，排序，实体参数查询等 * **注意: 使用的是数据库表字段，不是 Java 属性!** |

### 条件参数

|  |
| --- |
|  |

## 自定义SQL

### 注解方式

|  |
| --- |
|  |

### xml方式

|  |
| --- |
| 在mapper接口相同目录下创建Mapper.xml文件  <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"com.kazu.dao.UserMapper"*>  <select id=*"findAll"* resultType=*"com.kazu.entity.User"*>  select \* from user  </select>  </mapper> |

## 分页插件

### 编写分页插件配置类

|  |
| --- |
|  |

### Mapper接口

|  |
| --- |
|  |

### Mapper映射文件

|  |
| --- |
|  |

### 测试类

|  |
| --- |
|  |

## 代码生成器

### 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!-- MySQL驱动包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <!-- MyBatisPlus驱动包 -->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.1.1</version>  </dependency>  <!-- druid 数据库连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency>  <!-- 以下是代码生成器的jar依赖 -->  <!-- 代码生成器核心jar依赖 -->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-generator</artifactId>  <version>3.1.1</version>  </dependency>    <!-- 使用默认的velocity模板 -->  <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity-engine-core</artifactId>  <version>2.0</version>  </dependency>  <!-- sfl4j -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  </dependency> |

### 编写代码生成的配置类

|  |
| --- |
| **public** **class** TestMP {    **private** **static** String *author* ="KazuGin";//作者名称  **private** **static** String *outputDir* ="E:\\广州科苑\\课堂代码\\Y2T06\\mybatisplus02\\src\\main\\java";//生成的位置  **private** **static** String *driver* ="com.mysql.cj.jdbc.Driver";//驱动，注意版本  //连接路径,注意修改数据库名称  **private** **static** String *url* ="jdbc:mysql://localhost:3306/db\_mybatisplus?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC";  **private** **static** String *username* ="root";//数据库用户名  **private** **static** String *password* ="root";//数据库密码  **private** **static** String *tablePrefix* ="tbl\_";//数据库表的前缀，如tbl\_user  **private** **static** String [] *tables* = {"tbl\_employee","user"}; //生成的表  **private** **static** String *parentPackage* = "com.bdqn";//顶级包结构  **private** **static** String *dao* = "dao";//数据访问层包名称  **private** **static** String *service* = "service";//业务逻辑层包名称  **private** **static** String *entity* = "entity";//实体层包名称  **private** **static** String *controller* = "controller";//控制器层包名称  **private** **static** String *mapperxml* = "dao";//mapper映射文件包名称      /\*\*  \* 代码生成 示例代码  \*/  @Test  **public** **void** testGenerator() {  //1. 全局配置  GlobalConfig config = **new** GlobalConfig();  config.setAuthor(*author*) // 作者  .setOutputDir(*outputDir*) // 生成路径  .setFileOverride(**true**) // 文件覆盖  .setIdType(IdType.***AUTO***) // 主键策略  .setServiceName("%sService") // 设置生成的service接口的名字的首字母是否为I，加%s则不生成I  .setBaseResultMap(**true**) //映射文件中是否生成ResultMap配置  .setBaseColumnList(**true**); //生成通用sql字段    //2. 数据源配置  DataSourceConfig dsConfig = **new** DataSourceConfig();  dsConfig.setDbType(DbType.***MYSQL***) // 设置数据库类型  .setDriverName(*driver*) //设置驱动  .setUrl(*url*) //设置连接路径  .setUsername(*username*) //设置用户名  .setPassword(*password*); //设置密码    //3. 策略配置  StrategyConfig stConfig = **new** StrategyConfig();  stConfig.setCapitalMode(**true**) //全局大写命名  .setNaming(NamingStrategy.***underline\_to\_camel***) // 数据库表映射到实体的命名策略  .setTablePrefix(*tablePrefix*) //表前缀  .setInclude(*tables*); // 生成的表    //4. 包名策略配置  PackageConfig pkConfig = **new** PackageConfig();  pkConfig.setParent(*parentPackage*)//顶级包结构  .setMapper(*dao*) //数据访问层  .setService(*service*) //业务逻辑层  .setController(*controller*) //控制器  .setEntity(*entity*) //实体类  .setXml(*mapperxml*); //mapper映射文件    //5. 整合配置  AutoGenerator ag = **new** AutoGenerator();  ag.setGlobalConfig(config)  .setDataSource(dsConfig)  .setStrategy(stConfig)  .setPackageInfo(pkConfig);  //6. 执行  ag.execute();  }  } |

### 测试类

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** Mybatisplus02ApplicationTests {  @Resource  **private** EmployeeService employeeService;  @Test  **public** **void** test() {  List<Employee> list = employeeService.list();  **for** (Employee employee : list) {  System.***out***.println(employee);  }  }  } |

# 定时任务

|  |
| --- |
| @Component//标识改类为定时任务组件  **public** **class** TackTest {  /\*\*  \* 定义任务  \* fixedDelay：使用：@Scheduled(fixedDelay=3000)  \* 表示每隔多长时间发送一次、单位是毫秒  \*/  /\*@Scheduled(fixedDelay=3000)\*/  @Scheduled(cron="30 16 12 31 12 ?")//指定时间段来发送一次---cron表达式在百度上搜索cron在线表达式生成器匹配即可  **public** **void** sendEmail() {  System.***out***.println("=============定时任务=============");  }  } |
|  |

### 2:测试

在speingBoot项目的启动类上加上注解：@EnableScheduling//开启定时任务