MybatisPlus

MybatisPlus概述

MybatisPlus可以节省我们大量的工作时间，所有的CRUD代码他都可以自动化完成

Mybatis简介：

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。

特性：

- **无侵入**：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
- **损耗小**：启动即会自动注入基本 CURD，性能基本无损耗，直接面向对象操作
- **强大的 CRUD 操作**：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求
- **支持 Lambda 形式调用**：通过 Lambda 表达式，方便的编写各类查询条件，无需再担心字段写错
- **支持主键自动生成**：支持多达 4 种主键策略（内含分布式唯一 ID 生成器 - Sequence），可自由配置，完美解决主键问题
- **支持 ActiveRecord 模式**：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可进行强大的 CRUD 操作
- **支持自定义全局通用操作**：支持全局通用方法注入（ Write once, use anywhere ）
- **内置代码生成器**：采用代码或者 Maven 插件可快速生成 Mapper 、 Model 、 Service 、 Controller 层代码，支持模板引擎，更有超多自定义配置等您来使用
- **内置分页插件**：基于 MyBatis 物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同于普通 List 查询
- **分页插件支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer 等多种数据库
- **内置性能分析插件**：可输出 Sql 语句以及其执行时间，建议开发测试时启用该功能，能快速揪出慢查询
- **内置全局拦截插件**：提供全表 delete 、 update 操作智能分析阻断，也可自定义拦截规则，预防误操作

1、快速入门

借鉴官网的步骤

快速开始

我们将通过一个简单的 Demo 来阐述 MyBatis-Plus 的强大功能，在此之前，我们假设您已经：

- 拥有 Java 开发环境以及相应 IDE
- 熟悉 Spring Boot
- 熟悉 Maven

现有一张 `User` 表，其表结构如下：

id	name	age	email
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

1、创建数据库，并插入数据

Plain Text

```
DROP TABLE IF EXISTS user;
CREATE TABLE user
(
    id BIGINT(20) NOT NULL 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)
);


```

Plain Text

```
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');
```

2、初始化工程：添加依赖

尽量不要同时导入mybatis和mybatis-plus，版本的差异

JavaScript

```
<dependencies>
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>3.4.1</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.21</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </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>
</dependencies>
```

3、配置数据库连接的信息

数据库url要设置时区：serverTimezone=GMT%2B8

JavaScript

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/user?useSSL=true
&useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=admin
```

在 Spring Boot 启动类中添加 `@MapperScan` 注解，扫描 Mapper 文件夹：

Java

```java
@MapperScan("com.pge.mapper")
@SpringBootApplication
public class MybatisPlusApplication {

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

}
```

4、实体类和Mapper类

PHP

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

//在对应的Mapper上面继承基本的类BaseMapper,所有的CRUD操作都已经编写完成
@Repository //代表持久层
public interface UserMapper extends BaseMapper<User> {

}
```

2、配置日志

Shell

```
配置日志
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl
```

3、CRUD扩展

insert

Dockerfile

```
@Test
void contextLoads() {
    User user = new User();
    user.setAge(33);
    user.setEmail("2659093940@qq.com");
    user.setName("PGE");
    int insert = userMapper.insert(user);
    System.out.println(insert);
}

```

	id	name	age	email
1	1	Jone	18	test1@baomidou.com
2	2	Jack	20	test2@baomidou.com
3	3	Tom	28	test3@baomidou.com
4	4	Sandy	21	test4@baomidou.com
5	5	Billie	24	test5@baomidou.com
6	1339110985117478913	PGE	33	2659093940@qq.com

自动生成的id

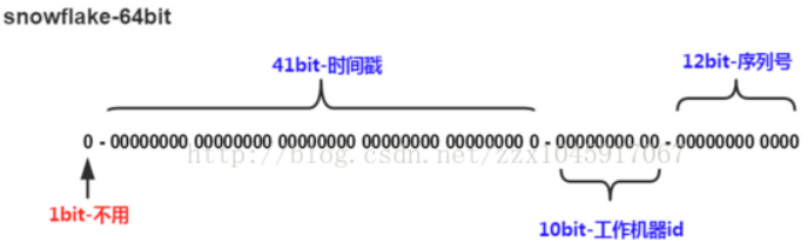
> 数据库插入的id的默认值：全局的唯一id

主键生成策略：

自3.3.0开始,默认使用**雪花算法+UUID**(不含中划线)

分布式系统唯一id：https://blog.csdn.net/zzx1045917067/article/details/78546902

雪花算法：



如图所示，这里第1位不可用，前41位表示时间，中间10位用来表示工作机器的id，后12位的序列号.

其中时间比较好理解，工作机器id则是机器标识，序列号是一个自增序列。有多少位表示在这一个单位时间内，此机器最多可以支持2^12个并发。在进入下一个时间单位后，序列号归0。

简单来说：就是把64的Long型数据由以下几个部分组成：

符号位(1位)-时间戳(41位)-数据中心标识(5位)-ID生成器实例标识(5位)-序列号(12位)

方法	主键生成策略	主键类型	说明
nextId	ASSIGN_ID，~~ID_WORKER~~，~~ID_WORKER_STR~~	Long,Integer,String	支持自动转换为String类型，但数值类型不支持自动转换，需精准匹配，例如返回Long，实体主键就不支持定义为Integer
nextUUID	ASSIGN_UUID，~~UUID~~	String	默认不含中划线的UUID生成

在实体类中设置主键生成策略

Java

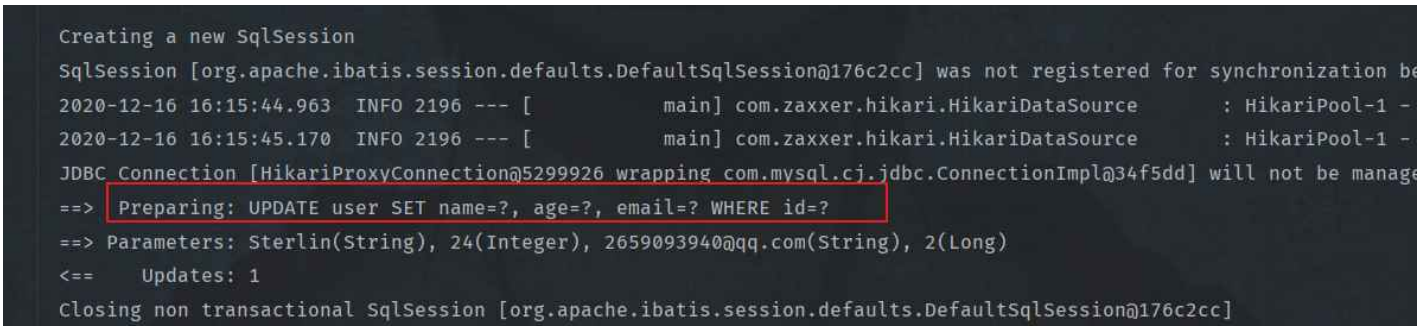
```java
@Data
public class User {
    //d对应数据库中的主键（uuid，自增id，雪花算法，redis，zookeeper）
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    private String name;
    private Integer age;
    private String email;
}

public enum IdType {
    AUTO(0), //数据库id自增
    NONE(1), //未设置主键
    INPUT(2), //手动输入
    ASSIGN_ID(3), // 全局唯一id
    ASSIGN_UUID(4), //默认的全局唯一id
    /** @deprecated */
    @Deprecated
    ID_WORKER(3),
    /** @deprecated */
    @Deprecated
    ID_WORKER_STR(3),
    /** @deprecated */
    @Deprecated
    UUID(4);
   }
```

update

Dockerfile

```
@Test
void contextLoads() {
    User user = new User();
    user.setId(2L);
    user.setAge(24);
    user.setEmail("2659093940@qq.com");
    user.setName("Sterlin");
    userMapper.updateById(user);

}
```

所有的sql都是自动帮你动态配置的



自动填充

创建时间、修改时间，这些操作一遍都是自动化生成的，我们不希望手动更新！
阿里巴巴开发手册：所有的数据库表：gmt_create(创建时间) 、gmt_modified(更新时间) 几乎所有的表都要配置上，而且需要自动化

> 方式一：数据库级别，在表上增加两个字段

```
`create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '数据创建时间',
`update_time` timestamp NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '数据更新时间'
```

> 方式二：代码级别

1、在实体类的字段上增加相应的注解

Java

```java
 //字段添加填充内容
@TableField(fill = FieldFill.INSERT)
private Date createTime;
@TableField(fill = FieldFill.INSERT_UPDATE)
private Date updateTime;
```

2、编写一个处理器来处理注解

自定义一个处理器，并继承MetaObjectHandler

Java

```java
@Component //不要忘记把处理器加到IOC容器中
public class MyMetaObjectHandler implements MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        //setFieldValByName(String fieldName, Object fieldVal, MetaObject metaObject)
        this.setFieldValByName("createTime",new Date(),metaObject);
        this.setFieldValByName("updateTime",new Date(),metaObject);
    }

    @Override
    public void updateFill(MetaObject metaObject) {

        this.setFieldValByName("updateTime",new Date(),metaObject);

    }
}
```

乐观锁

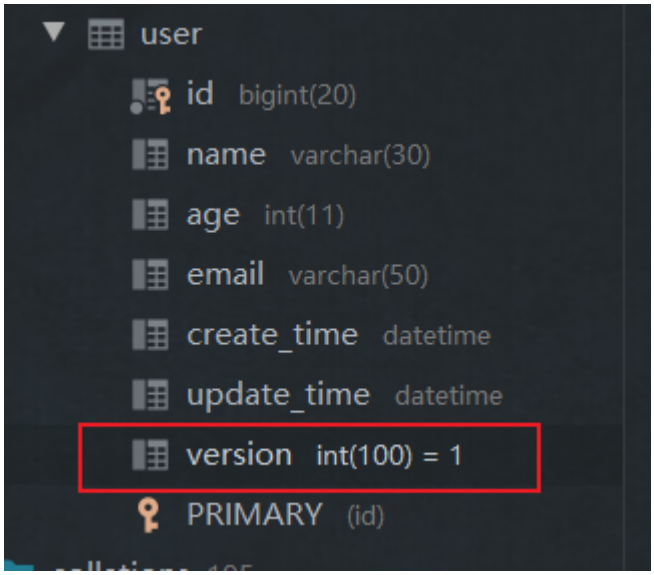
> 乐观锁：十分乐观，他总是认为不会出现问题，无论干什么都不会上锁，如果出现了问题，再次更新测试值
> 悲观锁：十分悲观，他总是认为会出问题，无论干什么都会上锁，再去操作

乐观锁的实现方式：

- 取出记录，获取当前version
- 更新时，带上这个version
- 执行更新时，比较这个version是否有无变化，set version = new version where version = oldVersion(相当于前面取出来的version)

MybatisPlus来实现乐观锁：

1、向数据库表中增加字段version，并在实体类中添加字段，而且在字段上配上注解



Java

```java
@Version
private Integer version;
```

2、自定义配置文件

> 新版本的 mybatisplus-plus 会出现这个问题,当我们根据官方文档使用乐观锁的相关代码时会出现这个问题:注意 OptimisticLockerInterceptor 不要写成 OptimisticLockerInnerInterceptor 就可以啦。

Java

```java
//扫描我们的mapper文件夹
@MapperScan("com.pge.mapper")
@EnableTransactionManagement
@Configuration //配置类
public class MybatisPlusConfig {

    //注册乐观锁插件
    @Bean
    public OptimisticLockerInterceptor optimisticLockerInterceptor(){
        return new OptimisticLockerInterceptor();
    }
}

```

3、测试

Dockerfile

```
@Test
void contextLoads() {

   User user = userMapper.selectById(2L);
   user.setName("Kobe");
   user.setEmail("kun@manchester.com");

   User user1 = userMapper.selectById(2L);
   user1.setName("James");
   user1.setEmail("kun@manchester.com");
   userMapper.updateById(user);
   userMapper.updateById(user1);

}
```

select

Java

```java
@Test
void contextLoads() {

    //单个查询
    User user = userMapper.selectById(2L);
    System.out.println(user);

    //多个id查询
    List<User> users = userMapper.selectBatchIds(Arrays.asList(1L, 2L, 3L, 4L));
    users.forEach(System.out::println);

    //map查询
    HashMap<String, Object> map = new HashMap<>();
    map.put("name","Kobe");
    List<User> users1 = userMapper.selectByMap(map);
    users1.forEach(System.out::println);

}
```

分页查询

MP其实内置了分页插件

使用：

1、配置分页插件

Java

```java
//扫描我们的mapper文件夹
@MapperScan("com.pge.mapper")
@EnableTransactionManagement
@Configuration //配置类
public class MybatisPlusConfig {

    //注册乐观锁插件
    @Bean
    public OptimisticLockerInterceptor optimisticLockerInterceptor(){
        return new OptimisticLockerInterceptor();
    }

     //配置分页插件
    @Bean
    public PaginationInterceptor paginationInterceptor(){
        return new PaginationInterceptor();
    }
}
```

2、测试

Java

```java
@Test
void contextLoads() {

   Page<User> page = new Page<>(2,3);
   userMapper.selectPage(page,null);

   page.getRecords().forEach(System.out::println);

}
```

delete

C++

```
@Test
void contextLoads() {

    //通过id删除
    userMapper.deleteById(1339118594583179265L);

    //通过批量删除
    userMapper.deleteBatchIds(Arrays.asList(1339110985117478913L,5L));

    //通过map删除
    HashMap<String, Object> map = new HashMap<>();
    map.put("name","Kobe");
    userMapper.deleteByMap(map);

}
```

逻辑删除

> 物理删除：从数据库中直接移除
> 逻辑删除：并没有在数据库中被移除，而是通过一个变量让他失效，例如 deleted=0 -> deleted=1

逻辑删除好比，管理员能查看被删除的数据，类似于回收站

使用：
配置`com.baomidou.mybatisplus.core.config.GlobalConfig$DbConfig`
步骤一：

SQL

```
mybatis-plus:
  global-config:
    db-config:
      logic-delete-field: flag  # 全局逻辑删除的实体字段名(since 3.3.0,配置后可以忽略不配置步骤2)
      logic-delete-value: 1 # 逻辑已删除值(默认为 1)
      logic-not-delete-value: 0 # 逻辑未删除值(默认为 0)
```

步骤二：在数据库增加字段deleted,并在实体类上增加注解@TableLogic

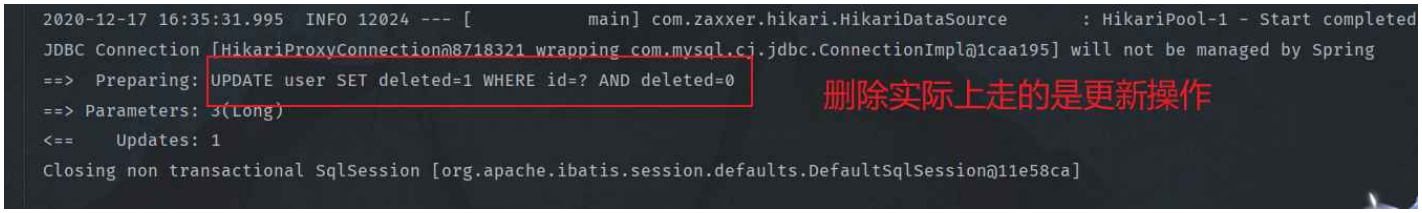
Java

```java
@TableLogic
private Integer deleted;
```

测试

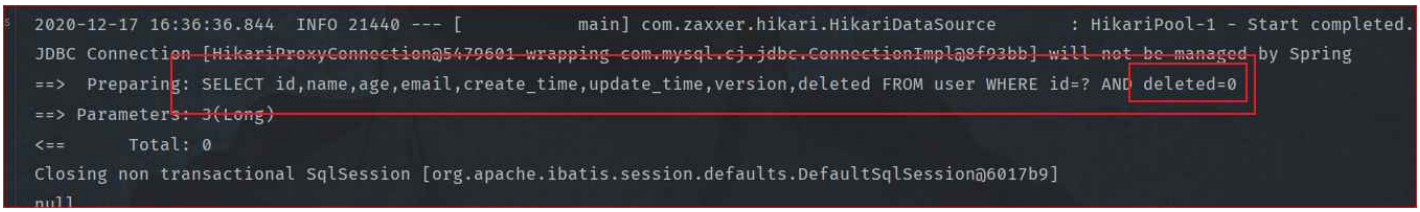
Java

```java

    @Test
    void contextLoads() {

        userMapper.deleteById(3L);

//    User user = userMapper.selectById(3L);
//    System.out.println(user);

    }
```



Java

```java
     @Test
    void contextLoads() {

//    userMapper.deleteById(3L);

        User user = userMapper.selectById(3L);
        System.out.println(user);

    }
```



4、性能分析插件

我们在平时的开发中，会遇到一些慢sql，我们需要测试

使用性能分析插件，可以帮助我们提高效率

作用：性能分析拦截器，用于输出每条 SQL 语句及其执行时间

MP也提供性能分析插件，如果超过这个时间就停止运行！

1、导入插件

```
/**
     * SQL执行效率插件
     */
@Bean
@Profile({"dev","test"})// 设置 dev test 环境开启，保证我们的效率
public PerformanceInterceptor performanceInterceptor() {
    PerformanceInterceptor performanceInterceptor = new PerformanceInterceptor();
    performanceInterceptor.setMaxTime(100); // ms设置sql执行的最大时间，如果超过了则不执行
    performanceInterceptor.setFormat(true); // 是否格式化代码
    return performanceInterceptor;
}
```

记住，要在SpringBoot中配置环境为dev或者 test 环境！

如果Mybatis-Plus版本在3.2.0以上，建议使用下面的执行SQL分析打印方式

执行SQL分析打印：

这个功能依赖p6spy组件，完美的输出打印SQL以及执行时长，支持MP3.1.0以上版本。

步骤一：导入依赖

JavaScript

```
<!-- https://mvnrepository.com/artifact/p6spy/p6spy -->
<dependency>
    <groupId>p6spy</groupId>
    <artifactId>p6spy</artifactId>
    <version>3.9.1</version>
</dependency>
```

步骤二：更改properties配置文件连接数据库配置，主要修改driver-class-name、url中jdbc后需要加上p6spy

Apache

```
配置环境为dev
spring.profiles.active=dev

spring.datasource.driver-class-name=com.p6spy.engine.spy.P6SpyDriver
spring.datasource.url=jdbc:p6spy:mysql://localhost:3306/mydatabase?useSSL=true&useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=admin
```

步骤三：新增spy.properties文件，内容如下，可以根据需求相对应的修改文件。

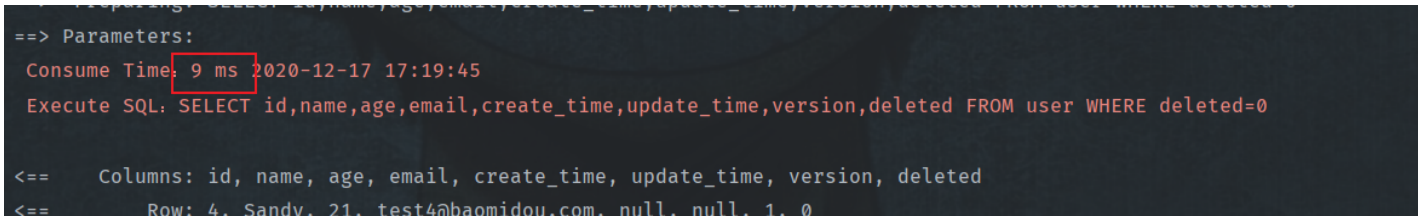
Makefile

```
module.log=com.p6spy.engine.logging.P6LogFactory,com.p6spy.engine.outage.P6OutageFactory
# 自定义日志打印
logMessageFormat=com.baomidou.mybatisplus.extension.p6spy.P6SpyLogger


#日志输出到控制台，解开注释就行了
# appender=com.baomidou.mybatisplus.extension.p6spy.StdoutLogger

# 指定输出文件位置
logfile=./src/com/pge/spy.log

# 使用日志系统记录 sql
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
# 设置 p6spy driver 代理
deregisterdrivers=true
# 取消JDBC URL前缀
useprefix=false
# 配置记录 Log 例外,可去掉的结果集有 error,info,batch,debug,statement,commit,rollback,result,resultset.
excludecategories=info,debug,result,batch,resultset
# 日期格式
dateformat=yyyy-MM-dd HH:mm:ss
# 实际驱动可多个
#driverlist=org.h2.Driver
# 是否开启慢SQL记录
outagedetection=true
# 慢SQL记录标准 2 秒
outagedetectioninterval=2
```

测试：

Java

```
@Test
void contextLoads() {
    userMapper.selectList(null);
}
```



5、条件构造器

测试：

PHP

```php
@Test
void contextLoads() {
    QueryWrapper<User> wrapper = new QueryWrapper<>();
    wrapper.like("name","J")
           .gt("age",30);
    List<User> users = userMapper.selectList(wrapper);
    users.forEach(System.out::println);
}

```

其他：参考官网 https://baomidou.com/guide/wrapper.html#alleq

6、代码自动生成器

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。
使用：参考官网