在application.properties配置文件中配置mybatis日志，用于输出执行的SQL语句
mybatis-plus.configuration.log-
impl=org.apache.ibatis.logging.stdout.StdOutImpl

如果想自定义mapper方法需要在resources中的mapper目录下的xxxMapper.xml文件中写SQL语句
若文件夹不是命名为mapper则需要在application.properties文件中进行如下配置
mybatis-plus.mapper-location=classpath:xxx/*.xml

项目或者模块名建议用 - 代替 _ 进行命名

雪花算法(snow flake)：由Twitter公布的分布式主键生成算法，能够保证不同表的主键的不重复性，以及相同表的主键的有序性；核心：长度64bit(一个long型)，1bit符号位0，41bit时间戳(毫秒级)，10bit机器标识ID，12bit毫秒内的流水号(序列号)；mybatisplus主键为id时，默认的主键策略 IdType.ASSIGN_ID (低版本中没有这个策略)，也就是雪花算法生成主键；如果主键不叫id，要使用@TableId指示主键是那个属性； 常使用在分库分表的情况。 保证主键既唯一，又有序。

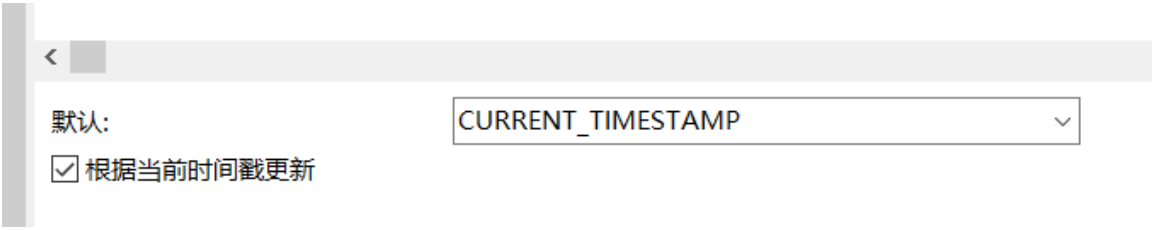
在application.properties中设置全局主键生成策略
mybatis-plus.global-config.db-config.id-type=AUTO

MyBatis-Plus 从 3.0.3 之后移除了代码生成器与模板引擎的默认依赖，需要手动添加相关依赖，才能实现代码生成器功能。
```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.2.0</version>
</dependency>
```

视图类主键属性不叫id需要使用@TableId(value="字段名") 指定属性

实体类属性名与字段完全不同时，需要在属性上使用@TableField(value="字段名")注解

一般数据库的create_time字段设置的默认值为CURRENT_TIMESTAMP，而update_time字段需要在设置同样默认值的前提下，勾选 根据当前时间戳更新 选项



也可以通过mybatisplus端设置自动填充 create_time 和 update_time

```
@TableField(fill = FieldFill.INSERT)
private LocalDateTime createTime;

@TableField(fill = FieldFill.INSERT_UPDATE)
private LocalDateTime updateTime;  // 时间类型最好使用这个数据类型
```

具体的填充逻辑：

```
@Component
@Slf4j  // lombok提供的日志功能
public class MyMetaObjectHandler implements MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        log.info("插入填充。。。");
        // 实现填充业务逻辑
        this.strictInsertFill(metaObject, "createTime", LocalDateTime.class,
LocalDateTime.now());  // (metaObject, "填充属性", 填充的数据类型, 填充的内容)
        this.strictInsertFill(metaObject, "updateTime", LocalDateTime.class,
LocalDateTime.now());  // (metaObject, "填充属性", 填充的数据类型, 填充的内容)

        // 如果业务层已经对age属性赋值，则不执行填充代码，执行判断
        Object age = this.getFieldValByName("age", metaObject);
        if (age == null){
            log.info("age 插入填充");
            this.strictInsertFill(metaObject, "age", Integer.class, 3);
        }

        // 每次填充之前，都应该判断一下当前对象是否包含这个属性
        boolean hasAuthor = metaObject.hasSetter("author");
        if (hasAuthor){
            log.info("author插入填充");
            this.strictInsertFill(metaObject, "author",String.class, "stone");
        }
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        log.info("更新填充。。。");
        this.strictInsertFill(metaObject, "updateTime", LocalDateTime.class,
LocalDateTime.now());
```

```
    }
}
```

spring boot环境下mybatisplus分页插件配置类：

```
package com.wxiang.mp.config;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.pagination.optimize.JsqlParserCountOptimize;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@MapperScan("scan.your.mapper.package")
public class MybatisPlusConfig {

    /** mybatis-plus 3.4.0版分页插件配置类
     * 新的分页插件,一缓和二缓遵循mybatis的规则,需要设置
MybatisConfiguration#useDeprecatedExecutor = false 避免缓存出现问题(该属性会在旧插件移
除后一同移除)
     */
    /*@Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.H2));
        return interceptor;

    }

    @Bean
    public ConfigurationCustomizer configurationCustomizer() {
        return configuration -> configuration.setUseDeprecatedExecutor(false);
    }*/


    // 3.4.0版本之前的分页插件配置
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new PaginationInterceptor();
        // 设置请求的页面大于最大页后操作， true调回到首页，false 继续请求  默认false
        paginationInterceptor.setOverflow(true);
        // 设置最大单页限制数量，默认 500 条，-1 不受限制
        paginationInterceptor.setLimit(20);
        // 开启 count 的 join 优化,只针对部分 left join
        paginationInterceptor.setCountSqlParser(new JsqlParserCountOptimize(true));
        return paginationInterceptor;
    }
```

}