林檬学院 JavaEE[www.edu.bjlemon.com] 讲师：梁辉 QQ:1170464456

Mybaits-Plus

一、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。

愿景

我们的愿景是成为 MyBatis 最好的搭档，就像 魂斗罗 中的 1P、2P，基友搭配，效率翻倍。

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
- **支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer2005、SQLServer 等多种数据库
- **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 XML 热加载**：Mapper 对应的 XML 支持热加载，对于简单的 CRUD 操作，甚至可以无 XML 启动
- **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 ）
- **支持关键词自动转义**：支持数据库关键词（order、key.....）自动转义，还可自定义关键词
- **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
- **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
- **内置 Sql 注入剥离器**：支持 Sql 注入剥离，有效预防 Sql 注入攻击

框架结构

文档资料

https://mp.baomidou.com

代码仓库

github:https://github.com/baomidou/mybatis-plus

gitee:https://gitee.com/baomidou/mybatis-plus

二、环境搭建

2.1 介绍

目前开发中使用的框架组合 通常为 SSM ,为了简化配置很多公司也使用了SpringBoot作为开发的脚手架，因此官网也提供了多种配置方式的例子，这里先演示基于传统的XML配置，SpringBoot配置更为简单，同学们可以自行查看官网的例子。

> 前提： 你需要掌握 Eclipse or IEDA的使用
>
> 你需要掌握 Maven 工具的使用
>
> 你需要掌握 Mybaits 框架的使用

如果你具备以上基础，那么可以接着学习MybaitsPlus了。

2.2 开始配置

准备数据数据：

```
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(60) DEFAULT NULL,
  `email` varchar(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of sys_user
-- ----------------------------
INSERT INTO `user` VALUES ('1', '刘德华', 'liudehua@qq.com');
INSERT INTO `user` VALUES ('2', '黎明', 'liming@qq.com');
INSERT INTO `user` VALUES ('3', '张学友', 'zxy@qq.com');
INSERT INTO `user` VALUES ('4', '郭富城', 'gfc@163.com');
```

编写 领域模型

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

> 这里使用了 lombok 工具，通过@Data注解自动生成 getts/setts/toString 方法。
>
> 如果没有请自行安装。

环境搭建和配置：

① pom.xml 文件

```
  <!--Junit 单元测试工具-->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>

        <!--Log4j 日志工具-->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>

        <!--MybatisPlus依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus</artifactId>
            <version>3.0.6</version>
        </dependency>

        <!--Spring IOC容器-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.2.RELEASE</version>
        </dependency>

        <!--Spring orm 支持-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>5.0.10.RELEASE</version>
        </dependency>

        <!--Mysql 数据库驱动-->
        <dependency>
```

```
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>

        <!--Druid数据源 连接池-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.10</version>
        </dependency>

        <!-- Lombok 工具-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.22</version>
            <scope>provided</scope>
        </dependency>
```

> 注意事项
>
> 引入 `MyBatis-Plus` 之后请不要再次引入 `MyBatis` 以及 `MyBatis-Spring`，以避免因版本差异导致的问题。

② db.properties 数据库连接配置文件

```
jdbc.user=root
jdbc.password=root
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8
```

③ log4j.properties 日志配置文件

```
# Global logging configuration
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

④ Spring 配置文件

```
<!--配置数据源-->
    <context:property-placeholder location="db.properties" />
    <bean name="dataSource"  class="com.alibaba.druid.pool.DruidDataSource">
             <property name="url" value="${jdbc.url}"/>
             <property name="driverClassName" value="${jdbc.driver}"/>
             <property name="username" value="${jdbc.user}"/>
             <property name="password" value="${jdbc.password}"/>
    </bean>
    <!--配置SqlSessionFactoryBean-->
```

```
    <bean name="sqlSessionFactory"

class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!--配置配置扫描器-->
    <bean name="scannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
          <property name="basePackage" value="com.bjlemon.mapper"/>
    </bean>
```

> 注意事项
>
> 整合 MybiatsPuls 框架 只需要调整 SqlSessionFactory 为 MyBatis-Plus 的 SqlSessionFactory 即可。

2.3 测试

```
ApplicationContext ioc = new ClassPathXmlApplicationContext("application.xml");

UserMapper userMapper = ioc.getBean(UserMapper.class, "userMapper");

User user= userMapper.selectById(1);

System.out.println(user);
----------------------------------------------------------------------------------------------


 _ _   |_  _ _|_. ___ _ |    _
| | |\/|_)(_| | |_\  |_)||_|_\
     /               |
                         3.0.6
 INFO [main] - {dataSource-1} inited
 User(id=1, name=刘德华, email=liudehua@qq.com)
 Process finished with exit code 0
```

三、MyBaitsPlus配置

3.0 、整合Mybaits

```
<!--基础配置-->
<property name="configLocation" value="mybatis-config.xml"/>
<property name="mapperLocations" value="classpath:mapper/*.xml"/>
<property name="typeAliasesPackage" value="com.bjlemon.domain"/>
```

3.1、Mybaits配置

```
<property name="configuration">
    <bean class="com.baomidou.mybatisplus.core.MybatisConfiguration">
        <property name="logImpl" value="org.apache.ibatis.logging.log4j.Log4jImpl"/>
        <property name="mapUnderscoreToCamelCase" value="true"/>
    </bean>
</property>
```

3.2、映射相关注解

@TableName

@TableField

@TableId

@TableLogic [后续讲解]

3.3、全局配置

```
 <!--全局配置-->
<property name="globalConfig">
    <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig">
        <!--数据库配置-->
        <property name="dbConfig">
            <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
                <property name="tablePrefix" value="sys_"/>
                <property name="idType" value="AUTO"/>
            </bean>
        </property>
    </bean>
</property>
```

3.4、分页插件配置

```
<property name="plugins">
    <array>
        <!-- 分页插件配置 -->
        <bean id="paginationInterceptor"
              class="com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor"/>
    </array>
</property>
```

四、通用CRUD

4.1 BaseMapper （通用Mapper）

insert

```
/**
 * <p>
 * 插入一条记录
 * </p>
 *
 * @param entity 实体对象
 * @return 插入成功记录数
 */
int insert(T entity);
```

deleteById

```
/**
 * <p>
 * 根据 ID 删除
 * </p>
 *
 * @param id 主键ID
 * @return 删除成功记录数
 */
int deleteById(Serializable id);
```

deleteByMap

```
/**
 * <p>
 * 根据 columnMap 条件，删除记录
 * </p>
 *
 * @param columnMap 表字段 map 对象
 * @return 删除成功记录数
 */
int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
```

deleteBatchIds

```
/**
 * <p>
 * 删除（根据ID 批量删除）
 * </p>
 *
 * @param idList 主键ID列表(不能为 null 以及 empty)
 * @return 删除成功记录数
 */
int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
```

updateById

```
/**
 * <p>
 * 根据 ID 修改
 * </p>
 *
 * @param entity 实体对象
 * @return 修改成功记录数
 */
int updateById(@Param(Constants.ENTITY) T entity);
```

selectById

```
/**
 * <p>
 * 根据 ID 查询
 * </p>
 *
 * @param id 主键ID
 * @return 实体
 */
T selectById(Serializable id);
```

selectList

```
/**
 * <p>
 * 根据 entity 条件，查询全部记录
 * </p>
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 实体集合
 */
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectBatchIds

```
/**
 * <p>
 * 查询（根据ID 批量查询）
 * </p>
 *
 * @param idList 主键ID列表(不能为 null 以及 empty)
 * @return 实体集合
 */
List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
```

selectByMap

```
/**
 * <p>
 * 查询 (根据 columnMap 条件)
 * </p>
 *
 * @param columnMap 表字段 map 对象
 * @return 实体集合
 */
List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
```

4.2 条件构造

前面的这些方法通常能够完成最基本的增删改查操作，但是要完成复杂的业务操作时，还是有局限性，比如我们需要通过 某些特定的条件 查询、修改 或者删除某些数据时，就不行了，在Mybaits 中可以使用标签 实现动态SQL 在MybatisPlus 中实现更为简单，MybatisPlus 提供了强大的条件构造工具，可以很轻松实现多条件查询。

条件构造接口和实现类

接口 Wrapper

抽象 AbstractWrapper

实现 QueryWrapper UpdateWrapper

实现 EmptyWrapper

查询方式	说明
setSqlSelect	设置 SELECT 查询字段
where	WHERE 语句，拼接 + `WHERE 条件`
and	AND 语句，拼接 + `AND 字段=值`
andNew	AND 语句，拼接 + `AND (字段=值)`
or	OR 语句，拼接 + `OR 字段=值`
orNew	OR 语句，拼接 + `OR (字段=值)`
eq	等于=
allEq	基于 map 内容等于=
ne	不等于<>
gt	大于>
ge	大于等于>=
lt	小于<
le	小于等于<=
like	模糊查询 LIKE
notLike	模糊查询 NOT LIKE
in	IN 查询
notIn	NOT IN 查询
isNull	NULL 值查询
isNotNull	IS NOT NULL
groupBy	分组 GROUP BY
having	HAVING 关键词
orderBy	排序 ORDER BY
orderAsc	ASC 排序 ORDER BY
orderDesc	DESC 排序 ORDER BY
exists	EXISTS 条件语句
notExists	NOT EXISTS 条件语句
between	BETWEEN 条件语句

查询方式	说明
notBetween	NOT BETWEEN 条件语句
addFilter	自由拼接 SQL
last	拼接在最后，例如：last("LIMIT 1")

delete

```
/**
 * <p>
 * 根据 entity 条件，删除记录
 * </p>
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 删除成功记录数
 */
int delete(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

update

```
/**
 * <p>
 * 根据 whereEntity 条件，更新记录
 * </p>
 *
 * @param entity        实体对象 (set 条件值,不能为 null)
 * @param updateWrapper 实体对象封装操作类（可以为 null,里面的 entity 用于生成 where 语句）
 * @return 修改成功记录数
 */
int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T>
updateWrapper);
```

selectOne

```
/**
 * <p>
 * 根据 entity 条件，查询一条记录
 * </p>
 *
 * @param queryWrapper 实体对象
 * @return 实体
 */
T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectCount

```
/**
 * <p>
 * 根据 Wrapper 条件，查询总记录数
 * </p>
 *
 * @param queryWrapper 实体对象
 * @return 满足条件记录数
 */
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectList

```
/**
 * <p>
 * 根据 entity 条件，查询全部记录
 * </p>
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 实体集合
 */
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectMaps

```
/**
 * <p>
 * 根据 Wrapper 条件，查询全部记录
 * </p>
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 字段映射对象 Map 集合
 */
List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectObjs

```
/**
 * <p>
 * 根据 Wrapper 条件，查询全部记录
 * 注意：  只返回第一个字段的值
 * </p>
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 字段映射对象集合
 */
List<Object> selectObjs(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectPage

```
/**
 * <p>
 * 根据 entity 条件，查询全部记录（并翻页）
 * </p>
 *
 * @param page         分页查询条件（可以为 RowBounds.DEFAULT）
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 * @return 实体分页对象
 */
IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

selectMapsPage

```
/**
 * <p>
 * 根据 Wrapper 条件，查询全部记录（并翻页）
 * </p>
 *
 * @param page         分页查询条件
 * @param queryWrapper 实体对象封装操作类
 * @return 字段映射对象 Map 分页对象
 */
IPage<Map<String, Object>> selectMapsPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T>
queryWrapper);
```

4.2 IService (通用Service)

> 说明:
>
> - 通用 Service CRUD 封装IService接口，进一步封装 CRUD 采用 `get 查询单行` `remove 删除`` `list 查询集合` `page 分页` 前缀命名方式区分 `Mapper` 层避免混淆，
> - 泛型 `T` 为任意实体对象
> - 建议如果存在自定义通用 Service 方法的可能，请创建自己的 `IBaseService` 继承 `Mybatis-Plus` 提供的基类
> - 对象 `Wrapper` 为 [条件构造器]
> - MyBaitsPlus 也基于IService 接口提供了实现 com.baomidou.mybatisplus.extension.service.impl.ServiceImplServiceImpl

```
public interface IService<T> {
    boolean save(T var1);
    default boolean saveBatch(Collection<T> entityList) {
        return this.saveBatch(entityList, 1000);
    }
    boolean saveBatch(Collection<T> var1, int var2);
    default boolean saveOrUpdateBatch(Collection<T> entityList) {
        return this.saveOrUpdateBatch(entityList, 1000);
    }

    boolean saveOrUpdateBatch(Collection<T> var1, int var2);
```

```
    boolean removeById(Serializable var1);
    boolean removeByMap(Map<String, Object> var1);
    boolean remove(Wrapper<T> var1);
    boolean removeByIds(Collection<? extends Serializable> var1);
    boolean updateById(T var1);
    boolean update(T var1, Wrapper<T> var2);
    default boolean updateBatchById(Collection<T> entityList) {
        return this.updateBatchById(entityList, 1000);
    }
    boolean updateBatchById(Collection<T> var1, int var2);
    boolean saveOrUpdate(T var1);
    T getById(Serializable var1);
    Collection<T> listByIds(Collection<? extends Serializable> var1);
    Collection<T> listByMap(Map<String, Object> var1);
    default T getOne(Wrapper<T> queryWrapper) {
        return this.getOne(queryWrapper, false);
    }
    T getOne(Wrapper<T> var1, boolean var2);
    Map<String, Object> getMap(Wrapper<T> var1);
    default Object getObj(Wrapper<T> queryWrapper) {
        return SqlHelper.getObject(this.listObjs(queryWrapper));
    }
    int count(Wrapper<T> var1);
    default int count() {
        return this.count(Wrappers.emptyWrapper());
    }
    List<T> list(Wrapper<T> var1);
    default List<T> list() {
        return this.list(Wrappers.emptyWrapper());
    }
    IPage<T> page(IPage<T> var1, Wrapper<T> var2);
    default IPage<T> page(IPage<T> page) {
        return this.page(page, Wrappers.emptyWrapper());
    }
    List<Map<String, Object>> listMaps(Wrapper<T> var1);
    default List<Map<String, Object>> listMaps() {
        return this.listMaps(Wrappers.emptyWrapper());
    }
    List<Object> listObjs(Wrapper<T> var1);
    default List<Object> listObjs() {
        return this.listObjs(Wrappers.emptyWrapper());
    }
    IPage<Map<String, Object>> pageMaps(IPage<T> var1, Wrapper<T> var2);
    default IPage<Map<String, Object>> pageMaps(IPage<T> page) {
        return this.pageMaps(page, Wrappers.emptyWrapper());
    }
}
```

用户自定义Service UserService extends IService

UserServiceImpl extnds ServiecImpl implments UserSerivce

五、ActiveRecord(活动记录)

Active Record(活动记录)，是一种领域模型模式，特点是一个模型类对应关系型数据库中的一个表，而模型类的一个实例对应表中的一行记录，它是ORM的一种实现。

开启 ActiveRecord 需要两个步骤

1、 继承 Model

2、 指定主键（mybaits3.x 不需要做，因为可以根据model推测）

六、代码生成器

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。

```
// 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");
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("jobob");
        gc.setOpen(false);
        mpg.setGlobalConfig(gc);

        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/ant?
useUnicode=true&useSSL=false&characterEncoding=utf8");
        // dsc.setSchemaName("public");
        dsc.setDriverName("com.mysql.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("密码");
        mpg.setDataSource(dsc);

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

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        strategy.setSuperEntityClass("com.baomidou.ant.common.BaseEntity");
        strategy.setEntityLombokModel(true);
        strategy.setRestControllerStyle(true);
        strategy.setSuperControllerClass("com.baomidou.ant.common.BaseController");
        strategy.setInclude(scanner("表名"));
        strategy.setSuperEntityColumns("id");
        strategy.setControllerMappingHyphenStyle(true);
        strategy.setTablePrefix(pc.getModuleName() + "_");
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }

}
```

七、插件扩展

分页插件

```
<property name="plugins">
            <array>
                <!-- 分页插件配置 -->
                <bean id="paginationInterceptor"
                      class="com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor"/>
            </array>

 </property>
```

性能分析插件

```
                <!--性能分析插件-->
                <bean class="com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor">
                    <property name="maxTime"  value="500"/>
                    <property name="format" value="true"/>
                </bean>
```

乐观锁插件

```
 <!-- 乐观锁插件 -->
                <bean
class="com.baomidou.mybatisplus.extension.plugins.OptimisticLockerInterceptor"/>
```

八、自定全局配置

逻辑删除，是一种软删除的技术，不从数据库中删除数据，只是在数据表中添加一个字段来维护数据的状态。

逻辑删除配置：

第一方面：

```
<property name="globalConfig">
            <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig">
                <property name="dbConfig">
                    <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
                        <property name="tablePrefix" value="sys_"/>
                        <property name="idType" value="AUTO"/>
                        <property name="tableUnderline" value="true"/>
                        <property name="logicDeleteValue" value="-1"/>
                        <property name="logicNotDeleteValue" value="1"/>
                    </bean>

                </property>
```

```
                <!--配置逻辑删除-->
                <property name="sqlInjector">
                     <bean
class="com.baomidou.mybatisplus.extension.injector.LogicSqlInjector"/>
                </property>
            </bean>
        ......
</property>
```

第二方面：

```
public class Employee extends Model<Employee> {

    private static final long serialVersionUID = 1L;

    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    private String empName;

    private String empJob;

    private BigDecimal empSalary;


    // 乐观锁需要的版本标识
    @Version
    private Integer version;


    @TableLogic
    private  Integer deleted;

    ......

}
```

数据库中也要添加一列 deleted 标识数据是否为删除状态

九、Ieda快速开发插件