04mybatis-plus的使用

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。

> 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、mybatis-plus环境搭建

Emp.java

```java
package cn.tulingxueyuan.bean;

import java.util.Date;

public class Emp {

    private Integer empno;
    private String eName;
    private String job;
    private Integer mgr;
    private Date hiredate;
    private Double sal;
    private Double comm;
    private Integer deptno;

    public Emp() {
    }

    public Integer getEmpno() {
        return empno;
    }

    public void setEmpno(Integer empno) {
        this.empno = empno;
    }

    public String geteName() {
        return eName;
    }

    public void seteName(String eName) {
        this.eName = eName;
    }

    public String getJob() {
        return job;
    }
```

```
38
39      public void setJob(String job) {
40          this.job = job;
41      }
42
43      public Integer getMgr() {
44          return mgr;
45      }
46
47      public void setMgr(Integer mgr) {
48          this.mgr = mgr;
49      }
50
51      public Date getHiredate() {
52          return hiredate;
53      }
54
55      public void setHiredate(Date hiredate) {
56          this.hiredate = hiredate;
57      }
58
59      public Double getSal() {
60          return sal;
61      }
62
63      public void setSal(Double sal) {
64          this.sal = sal;
65      }
66
67      public Double getComm() {
68          return comm;
69      }
70
71      public void setComm(Double comm) {
72          this.comm = comm;
73      }
74
75      public Integer getDeptno() {
76          return deptno;
```

```
    }

    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }

    @Override
    public String toString() {
        return "Emp{" +
                "empno=" + empno +
                ", ename='" + eName + '\'' +
                ", job='" + job + '\'' +
                ", mgr=" + mgr +
                ", hiredate=" + hiredate +
                ", sal=" + sal +
                ", comm=" + comm +
                ", deptno=" + deptno +
                '}';
    }
}
```

数据库表sql语句

```
CREATE TABLE `tbl_emp` (
`EMPNO` int(4) NOT NULL AUTO_INCREMENT,
`E_NAME` varchar(10) DEFAULT NULL,
`JOB` varchar(9) DEFAULT NULL,
`MGR` int(4) DEFAULT NULL,
`HIREDATE` date DEFAULT NULL,
`SAL` double(7,2) DEFAULT NULL,
`COMM` double(7,2) DEFAULT NULL,
`DEPTNO` int(4) DEFAULT NULL,
PRIMARY KEY (`EMPNO`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

pom.xml

```
<?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/xs
```

```
    <modelVersion>4.0.0</modelVersion>

    <groupId>cn.tulingxueyuan</groupId>
    <artifactId>mybatis_plus</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/com.baomidou/mybatis-plus -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus</artifactId>
            <version>3.3.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/log4j/log4j -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.alibaba/druid -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.21</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.19</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
        <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.2.3.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>5.2.3.RELEASE</version>
        </dependency>

    </dependencies>

</project>
```

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
       PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
       <settings>
       <setting name="logImpl" value="LOG4J"/>
   </settings>
</configuration>
```

log4j.properties

```
# 全局日志配置
log4j.rootLogger=INFO, stdout
# MyBatis 日志配置
log4j.logger.cn.tulingxueyuan=truce
# 控制台输出
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

db.properties

```
driverClassname=com.mysql.jdbc.Driver
username=root
password=123456
url=jdbc:mysql://localhost:3306/demo
```

spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf
    <context:property-placeholder location="classpath:db.properties"></context:property-p
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${driverClassname}"></property>
        <property name="url" value="${url}"></property>
        <property name="username" value="${username}"></property>
        <property name="password" value="${password}"></property>
    </bean>
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTr
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager"></tx:annotation-driver
    <bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <property name="configLocation" value="classpath:mybatis-config.xml"></property>
    </bean>
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.tulingxueyuan.dao"></property>
    </bean>
</beans>
```

MyTest.java

```
package cn.tulingxueyuan;


```

```
import com.alibaba.druid.pool.DruidDataSource;
```

```java
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.sql.SQLException;

public class MyTest {

    ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

    @Test
    public void test01() throws SQLException {
        DruidDataSource dataSource = context.getBean("dataSource", DruidDataSource.class
        System.out.println(dataSource.getConnection());
    }
}
```

在集成mybatis-plus的时候非常简单，只需要替换mybatis自己的sqlSessionFactoryBean对象即可

```xml
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.extension.spring.Mybatis
        <property name="dataSource" ref="dataSource"></property>
        <property name="configLocation" value="classpath:mybatis-config.xml"></property>
        <property name="typeAliasesPackage" value="cn.tulingxueyuan.bean"></property>
    </bean>
```

2、简单的CRUD操作

如果我们下面要实现CRUD的基本操作，那么我们该如何实现呢?

在Mybatis中，我们需要编写对应的Dao接口，并在接口中定义相关的方法，然后提供与该接口相同名称的Dao.xml文件，在文件中填写对应的sql语句，才能完成对应的操作

在Mybatis-plus中，我们只需要定义接口，然后继承BaseMapper<T>类即可，此前做的所有操作都是由Mybatis-plus来帮我们完成，不需要创建sql映射文件

EmpDao.java

```java
package cn.tulingxueyuan.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import cn.tulingxueyuan.bean.Emp;

/**
```

```
 * 在mybatis操作的时候，我们需要自己定义接口中实现的方法，并添加与之对应的EmpDao.xml文件，编写对
 * 在mybatis-plus操作的时候，我们只需要继承BaseMapper接口即可，其中的泛型T表示我们要实际操作的实
 */
public interface EmpDao extends BaseMapper<Emp> {
}
```

1、插入操作

MyTest.java

```
package cn.tulingxueyuan;


import com.alibaba.druid.pool.DruidDataSource;
import cn.tulingxueyuan.bean.Emp;
import cn.tulingxueyuan.dao.EmpDao;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.sql.SQLException;
import java.util.Date;

public class MyTest {

    ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

    private EmpDao empDao = context.getBean("empDao",EmpDao.class);

@Test
    public void testInsert(){
        Emp emp = new Emp();
        emp.seteName("zhangsan");
        emp.setJob("Teacher");
        emp.setMgr(100);
        emp.setSal(1000.0);
        emp.setComm(500.0);
        emp.setHiredate(new Date());
        emp.setDeptno(10);
        int insert = empDao.insert(emp);
```

```
31          System.out.println(insert);
32      }
33  }
```

当运行上述代码的时候，大家发现报错了，原因在于你写的实体类的名称跟表的名称不匹配，因此在实现的是需要添加@TableName注解，指定具体的表的名称

```
1  @TableName("emp")
2  public class Emp {//省略内容}
```

上述代码运行通过之后，大家会发现结果能够正常的进行插入，但是在控制台会打印一个警告信息，说没有@TableId的注解，原因就在于定义实体类的时候并没有声明其中的主键是哪个列，以及使用什么样的主键生成策略，因此，可以在类的属性上添加如下注解，来消除此警告

```
1  public class Emp {
2
3      @TableId(value = "empno",type = IdType.AUTO)
4      private Integer empno;
5      private String eName;
6      private String job;
7      private Integer mgr;
8      private Date hiredate;
9      private Double sal;
10     private Double comm;
11     private Integer deptno;
12    //set、get、tostring方法省略
13 }
```

但是大家会发现，我们在写属性的时候，实体类属性名称跟表的属性名称并没有一一对应上，那么为什么会完成对应的操作呢?
其实原因就在于mybatis-plus的全局配置

在进行数据插入的是，如果我们输入的时候用的是全字段，那么sql语句中就会执行如下sql语句：
INSERT INTO tbl_emp (e_name, job, mgr, hiredate, sal, comm, deptno) VALUES (?, ?, ?, ?, ?, ?, ?)
但是如果我们在插入的时候，将对象中的某些属性值设置为空，那么会是什么效果呢?

```
1   @Test
2     public void testInsert(){
3         Emp emp = new Emp();
```

```
        emp.seteName("zhangsan");
        emp.setJob("Teacher");
        emp.setMgr(100);
//        emp.setSal(1000.0);
//        emp.setComm(500.0);
//        emp.setHiredate(new Date());
//        emp.setDeptno(10);
        int insert = empDao.insert(emp);
        System.out.println(insert);
        System.out.println(emp.getEmpno());
    }
```

INSERT INTO tbl_emp (e_name, job, mgr) VALUES (?, ?, ?)

大家其实可以看到我们在插入的时候，mybatis-plus会根据我会输入的对象的字段的个数来动态的调整我们的sql语句插入的字段，这是大家需要注意的mybatis-plus比较灵活的地方。

2、更新操作

```
@Test
    public void testUpdate(){
        Emp emp = new Emp();
        emp.setEmpno(1);
        emp.seteName("lisi");
        emp.setJob("student");
        emp.setMgr(100);
        emp.setSal(1000.0);
        emp.setComm(500.0);
        emp.setHiredate(new Date());
        emp.setDeptno(10);
        int update = empDao.updateById(emp);
        System.out.println(update);
    }
```

3、删除操作

```
    @Test
    public void testDelete(){
        // 1、根据id删除数据
//        int i = empDao.deleteById(1);
//        System.out.println(i);

        // 2、根据一组id删除数据
```

```
//        int i = empDao.deleteBatchIds(Arrays.asList(2, 3, 4));
//        System.out.println(i);

        // 3、根据条件删除数据
//        QueryWrapper queryWrapper = new QueryWrapper();
//        queryWrapper.in("empno",Arrays.asList(5,6,7));
//        int delete = empDao.delete(queryWrapper);
//        System.out.println(delete);

        // 4、条件封装map删除数据
        Map<String,Object> map = new HashMap<>();
        map.put("empno",9);
        int i = empDao.deleteByMap(map);
        System.out.println(i);
    }
```

4、查询操作

```
    @Test
    public void testselect(){

        // 1、根据id查询对象
//        Emp emp = empDao.selectById(1);
//        System.out.println(emp);

        // 2、根据实体包装类查询单个对象，返回的结果集有且仅能有一个对象
//        QueryWrapper<Emp> emp = new QueryWrapper<Emp>();
//        emp.eq("empno",2).eq("e_name","zhangsan");
//        Emp emp1 = empDao.selectOne(emp);
//        System.out.println(emp1);

        // 3、通过多个id值进行查询
//        List<Emp> list = empDao.selectBatchIds(Arrays.asList(1, 2, 3));
//        for (Emp emp : list) {
//            System.out.println(emp);
//        }

        // 4、通过map封装进行条件查询
//        Map<String,Object> map = new HashMap<String, Object>();
//        map.put("e_name","zhangsan");
```

```
23 //        map.put("sal",1000.0);
24 //        List<Emp> list = empDao.selectByMap(map);
25 //        for (Emp emp : list) {
26 //            System.out.println(emp);
27 //        }
28
29         // 5、分页查询,需要添加分页插件
30         /**
31          * <property name="plugins">
32          *             <array>
33          *                 <bean class="com.baomidou.mybatisplus.extension.plugins.Pagina
34          *             </array>
35          *         </property>
36          */
37
38        // Page<Emp> empPage = empDao.selectPage(new Page<>(2, 5), null);
39        // List<Emp> records = empPage.getRecords();
40        // System.out.println(records);
41
42                // 6、根据条件返回查询结果总数
43 //        QueryWrapper<Emp> queryWrapper = new QueryWrapper<>();
44 //        queryWrapper.eq("e_name","zhangsan");
45 //        Integer integer = empDao.selectCount(queryWrapper);
46 //        System.out.println(integer);
47
48         // 7、根据条件查询所有结果返回list集合
49 //        List<Emp> list = empDao.selectList(null);
50 //        for (Emp emp : list) {
51 //            System.out.println(emp);
52 //        }
53
54         // 8、根据条件查询结果封装成map的list结构
55 //        List<Map<String, Object>> maps = empDao.selectMaps(null);
56 //        System.out.println(maps);
57     }
```

3、Mybatis-plus的相关配置

在mybatis中我们可以在mybatis-config配置文件中可以添加<settings>标签，设置全局的默认策略，在MP中也具备相同的功能，只不过配置方式有所不同，我们可以在spring.xml文件中添加配置。

https://mp.baomidou.com/config/

在此链接中包含了非常多的配置项，用户可以按照自己的需求添加需要的配置，配置方式如下：

spring.xml

```
<bean id="sqlSessionFactory" class="com.baomidou.mybatisplus.extension.spring.MybatisSqlS
    <property name="configuration" ref="configuration"/> <!--  非必须  -->
    <property name="globalConfig" ref="globalConfig"/> <!--  非必须  -->
    ......
</bean>

<bean id="configuration" class="com.baomidou.mybatisplus.core.MybatisConfiguration">
    ......
</bean>

<bean id="globalConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig">
    <property name="dbConfig" ref="dbConfig"/> <!--  非必须  -->
    ......
</bean>

<bean id="dbConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
    ......
</bean>
```

通过这个配置文件的配置，大家可以进行回想上述问题的出现，mybatis-plus是如何解决这个问题的呢?

在mybatis-plus中会引入写默认的配置，这个选项的默认配置为true，因此可以完成对应的实现。

我们可以通过如下配置来禁用驼峰标识的操作，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spring
    <context:property-placeholder location="classpath:db.properties"></context:property-p
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${driverClassname}"></property>
        <property name="url" value="${url}"></property>
        <property name="username" value="${username}"></property>
        <property name="password" value="${password}"></property>
    </bean>
```

```
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTr
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager"></tx:annotation-driver
    <bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.extension.spring.MyB
        <property name="configLocation" value="classpath:mybatis-config.xml"></property>
        <property name="dataSource" ref="dataSource"></property>
        <property name="typeAliasesPackage" value="cn.tulingxueyuan.bean"></property>
        <property name="globalConfig" ref="globalConfig"></property>
        <property name="configuration" ref="configuration"></property>
    </bean>
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.tulingxueyuan.dao"></property>
    </bean>
    <bean id="configuration" class="com.baomidou.mybatisplus.core.MybatisConfiguration">
        <property name="mapUnderscoreToCamelCase" value="false"></property>
    </bean>

    <bean id="globalConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig">
        <property name="dbConfig" ref="dbConfig"></property>
    </bean>
    <bean id="dbConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig
    </bean>
</beans>
```

1、当添加上述配置之后，大家发现运行过程中报错，

Property 'configuration' and 'configLocation' can not specified with together

表示这两个标签无法同时使用，因此我们可以选择将configLocation给禁用掉，就是不使用mybatis的配置，此时就能够正常使用了，但是放置属性的时候又报错了，原因就在于我们把驼峰标识给禁用了，重新开启即可。除此之外，我们还可以在属性的上面添加@TableField属性

```
    @TableField(value = "e_name")
    private String eName;
```

2、此时发现日志功能又无法使用了，只需要添加如下配置即可

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

```
</bean>
```

3、我们在刚刚插入数据的时候发现每个类可能都需要写主键生成策略，这是比较麻烦的，因此可以选择将主键配置策略设置到全局配置中。

```
<bean id="dbConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
        <property name="idType" ref="idType"></property>
    </bean>
    <util:constant id="idType" static-field="com.baomidou.mybatisplus.annotation.IdType.A
```

4、如果你的表的名字都具备相同的前缀，那么可以设置默认的前缀配置策略，此时的话可以将实体类上的@TableName标签省略不写

```
<bean id="dbConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
        <property name="idType" ref="idType"></property>
        <property name="tablePrefix" value="tbl_"></property>
    </bean>
    <util:constant id="idType" static-field="com.baomidou.mybatisplus.annotation.IdType.A
```

5、在mybatis-plus中如果需要获取插入的数据的主键的值，那么直接获取即可，原因就在于配置文件中指定了默认的属性为true

4、条件构造器Wrapper（看官网即可）

5、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。

其实在学习mybatis的时候我们就使用过逆向工程，根据我们的数据表来生成的对应的实体类，DAO接口和Mapper映射文件，而MyBatis-plus提供了更加完善的功能，下面来针对两种方式做一个基本的对比

1、MyBatis-plus是根据java代码开生成代码的，而Mybatis是根据XML文件的配置来生成的

2、MyBatis-plus能够生成实体类、Mapper接口、Mapper映射文件，Service层，Controller层，而Mybatis只能生成实体类，Mapper接口，Mapper映射文件

1、操作步骤：

1、添加依赖

添加代码生成器依赖

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

```
    <version>3.3.1.tmp</version>
</dependency>
```

添加 模板引擎 依赖，MyBatis-Plus 支持 Velocity（默认）、Freemarker、Beetl，用户可以选择自己熟悉的模板引擎，如果都不满足您的要求，可以采用自定义模板引擎。

```
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-engine-core</artifactId>
    <version>2.2</version>
</dependency>
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.30</version>
</dependency>
<dependency>
    <groupId>com.ibeetl</groupId>
    <artifactId>beetl</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
```

2、编写生成类

```
import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
import com.baomidou.mybatisplus.generator.config.GlobalConfig;
import com.baomidou.mybatisplus.generator.config.PackageConfig;
import com.baomidou.mybatisplus.generator.config.StrategyConfig;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import org.junit.Test;

public class MyTest {



    @Test
    public void testGenerator(){
        //此处默认有两个对应的实现类，大家不要导错包
        GlobalConfig globalConfig = new GlobalConfig();
        //设置全局的配置
```

```
        globalConfig.setActiveRecord(true)//是否支持AR模式
                .setAuthor("lian")//设置作者
                .setOutputDir("e:\\self_project\\mybatisplus_generatorcode\\src\\main\\ja
                .setFileOverride(true)//设置文件覆盖
                .setIdType(IdType.AUTO) //设置主键生成策略
                .setServiceName("%sService")//设置生成的serivce接口的名字
                .setBaseResultMap(true) //设置基本的结果集映射
                .setBaseColumnList(true);//设置基本的列集合

        //设置数据源的配置
        DataSourceConfig dataSourceConfig = new DataSourceConfig();
        dataSourceConfig.setDriverName("com.mysql.cj.jdbc.Driver")
                .setUrl("jdbc:mysql://192.168.85.111:3306/mp?serverTimezone=UTC")
                .setUsername("root").setPassword("123456");

        // 进行策略配置
        StrategyConfig strategyConfig = new StrategyConfig();
        strategyConfig.setCapitalMode(true)//设置全局大写命名
                .setNaming(NamingStrategy.underline_to_camel)//数据库表映射到实体的命名策略
                .setTablePrefix("tbl_")//设置表名前缀
                .setInclude("tbl_emp");//生成的表

        // 进行包名的策略配置
        PackageConfig packageConfig = new PackageConfig();
        packageConfig.setParent("cn.tulingxueyuan")
                .setMapper("mapper")
                .setService("service")
                .setController("controller")
                .setEntity("bean")
                .setXml("mapper");

        //整合配置
        AutoGenerator autoGenerator = new AutoGenerator();
        autoGenerator.setGlobalConfig(globalConfig).setDataSource(dataSourceConfig).setSt
                .setPackageInfo(packageConfig);

        autoGenerator.execute();
    }
}
```

注意，当通过上述代码实现之后，大家发现可以在Controller层可以直接实现调用，这些调用的实现最核心的功能就在于ServiceImpl类，这个类中自动完成mapper的注入，同时提供了一系列CRUD的方法。

6、插件扩展

MyBatis 允许你在映射语句执行过程中的某一点进行拦截调用。默认情况下，MyBatis 允许使用插件来拦截的方法调用包括：

- Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
- ParameterHandler (getParameterObject, setParameters)
- ResultSetHandler (handleResultSets, handleOutputParameters)
- StatementHandler (prepare, parameterize, batch, update, query)

1、分页插件

在spring.xml文件中添加如下配置引入插件

```
<property name="plugins">
            <array>
                <bean class="com.baomidou.mybatisplus.extension.plugins.PaginationInterce
            </array>
        </property>
```

编写测试类

```
    @Test
    public void TestPage(){
        Page page = new Page(2,2);
        Page page1 = empDao.selectPage(page, null);
        List records = page1.getRecords();
        for (Object record : records) {
            System.out.println(record);
        }
        System.out.println("==============");
        System.out.println("获取总条数："+page.getTotal());
        System.out.println("当前页码："+page.getCurrent());
        System.out.println("总页码："+page.getPages());
        System.out.println("每页显示的条数："+page.getSize());
        System.out.println("是否有上一页："+page.hasPrevious());
        System.out.println("是否有下一页："+page.hasNext());
    }
```

2、乐观锁插件

当要更新一条记录的时候，希望这条记录没有被别人更新

乐观锁实现方式：

取出记录时，获取当前version 更新时，带上这个version 执行更新时， set version = newVersion where version = oldVersion 如果version不对，就更新失败

添加配置：

```
<bean class="com.baomidou.mybatisplus.extension.plugins.OptimisticLockerInterceptor"></b
```

修改实体类添加version字段并在表中添加version字段

编写测试类

```
@Test
    public void testOptimisticLocker(){
        Emp emp = new Emp();
        emp.setEmpno(22);
        emp.seteName("zhang");
        emp.setSal(10000.0);
        emp.setComm(1000.0);
        emp.setVersion(2);
        empDao.updateById(emp);
    }
```

3、SQL执行分析插件，避免出现全表更新和删除

```
<bean class="com.baomidou.mybatisplus.extension.plugins.SqlExplainInterceptor">
                    <property name="sqlParserList">
                        <list>
                            <bean class="com.baomidou.mybatisplus.extension.parsers.Bloc
                        </list>
                    </property>
                </bean>
```

```
@Test
public void testSqlExplain(){
    int delete = empDao.delete(null);
    System.out.println(delete);
}
```

4、非法sql检查插件

```
<bean class="com.baomidou.mybatisplus.extension.plugins.IllegalSQLInterceptor">
</bean>
```

```
@Test
public void testSqlIllegal(){
    QueryWrapper<Emp> queryWrapper = new QueryWrapper<>();
    queryWrapper.or();
    List<Emp> list = empDao.selectList(queryWrapper);
    for (Emp emp : list) {
        System.out.println(emp);
    }
}
```

7、SQL注入器

全局配置 `sqlInjector` 用于注入 `ISqlInjector` 接口的子类，实现自定义方法注入。也就是说我们可以将配置在xml中的文件使用注入的方式注入到全局中，就不需要再编写sql语句

自定义注入器

```
package cn.tulingxueyuan.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.injector.AbstractSqlInjector;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class MyInjector extends  AbstractSqlInjector{

    @Override
    public List<AbstractMethod> getMethodList(Class<?> mapperClass) {
        return Stream.of(new DeleteAll()).collect(Collectors.toList());
    }
}
```

添加配置：

```
 <bean id="globalConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig">
        <property name="dbConfig" ref="dbConfig"></property>
        <property name="sqlInjector" ref="myinject"></property>
    </bean>
    <bean id="myinject" class="cn.tulingxueyuan.injector.MyInjector"></bean>
```

```
package cn.tulingxueyuan.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.metadata.TableInfo;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.SqlSource;

public class DeleteAll extends AbstractMethod {
    @Override
    public MappedStatement injectMappedStatement(Class<?> mapperClass, Class<?> modelCla
        String sql;
        MySqlMethod mySqlMethod = MySqlMethod.DELETE_ALL;
        if (tableInfo.isLogicDelete()) {
            sql = String.format(mySqlMethod.getSql(), tableInfo.getTableName(),  tableIn
                    sqlWhereEntityWrapper(true,tableInfo));
        } else {
            mySqlMethod = MySqlMethod.DELETE_ALL;
            sql = String.format(mySqlMethod.getSql(), tableInfo.getTableName(),
                    sqlWhereEntityWrapper(true,tableInfo));
        }
        SqlSource sqlSource = languageDriver.createSqlSource(configuration, sql, modelCl
        return addUpdateMappedStatement(mapperClass, modelClass, mySqlMethod.getMethod()
    }
}
package cn.tulingxueyuan.injector;


   /**
    * 自定义全局删除方法
    */

    public enum MySqlMethod {

```

```
34
35      /**
36       * 删除全部
37       */
38      DELETE_ALL("deleteAll", "根据 entity 条件删除记录", "<script>\nDELETE FROM %s %s\n</sc
39
40
41      private final String method;
42      private final String desc;
43      private final String sql;
44
45      MySqlMethod(String method, String desc, String sql) {
46          this.method = method;
47          this.desc = desc;
48          this.sql = sql;
49      }
50
51      public String getMethod() {
52          return method;
53      }
54
55      public String getDesc() {
56          return desc;
57      }
58
59      public String getSql() {
60          return sql;
61      }
62
63  }
64  package cn.tulingxueyuan.dao;
65
66  import com.baomidou.mybatisplus.core.mapper.BaseMapper;
67  import cn.tulingxueyuan.bean.Emp;
68
69  /**
70   * 在mybatis操作的时候，我们需要自己定义接口中实现的方法，并添加与之对应的EmpDao.xml文件，编写对
71   * 在mybatis-plus操作的时候，我们只需要继承BaseMapper接口即可，其中的泛型T表示我们要实际操作的实
72   */
```

```
73 public interface EmpDao extends BaseMapper<Emp> {
74     Integer deleteAll();
75 }
```

8、公共字段填充

- 实现元对象处理器接口：com.baomidou.mybatisplus.core.handlers.MetaObjectHandler
- 注解填充字段 `@TableField(.. fill = FieldFill.INSERT)` 生成器策略部分也可以配置！

metaobject:元对象，是mybatis提供的一个用于更加方便，更加优雅的访问对象的属性，给对象的属性设置值的一个对象，还会用于包装对象，支持Object,Map,Collection等对象进行包装。本质上metaobject是给对象的属性设置值，最终还是要通过Reflect获取到属性的对应方法的invoker，最终执行。

编写自定义的公共字段填充

```
1  package cn.tulingxueyuan.fill;
2
3  import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
4  import org.apache.ibatis.reflection.MetaObject;
5
6  import java.time.LocalDateTime;
7  import java.util.stream.Stream;
8
9  public class MyMetaObjectHandler implements MetaObjectHandler {
10
11     @Override
12     public void insertFill(MetaObject metaObject) {
13         this.strictInsertFill(metaObject, "eName", String.class, "lian"); // 起始版本 3.3
14 //        this.fillStrategy(metaObject, "createTime", LocalDateTime.now()); // 也可以使用
15     }
16
17     @Override
18     public void updateFill(MetaObject metaObject) {
19         this.strictUpdateFill(metaObject, "eName", String.class,"lian"); // 起始版本 3.3.
20 //        this.fillStrategy(metaObject, "updateTime", LocalDateTime.now()); // 也可以使用
21     }
22 }
```

添加到对应的配置中：

```
1    <bean id="globalConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig">
```

```
        <property name="dbConfig" ref="dbConfig"></property>
        <property name="metaObjectHandler" ref="myMeta"></property>
    </bean>
    <bean id="myMeta" class="cn.tulingxueyuan.fill.MyMetaObjectHandler"></bean>
```

测试：

```
@Test
  public void testMeta(){
      int insert = empDao.insert(new Emp());
      System.out.println(insert);
 }
```