一、Mybatis-plus简介

1.1 定义

MyBtais-plus（简称MP），是MyBatis的增强工具包，只做增强不做改变，为了简化开发工作、提高生产力而生，现在MyBatis-plus已经更新到3.x版本了，本次以2.x版本记录使用方法。

1.2 官网

http://mp.baomidou.comm

1.3 特性

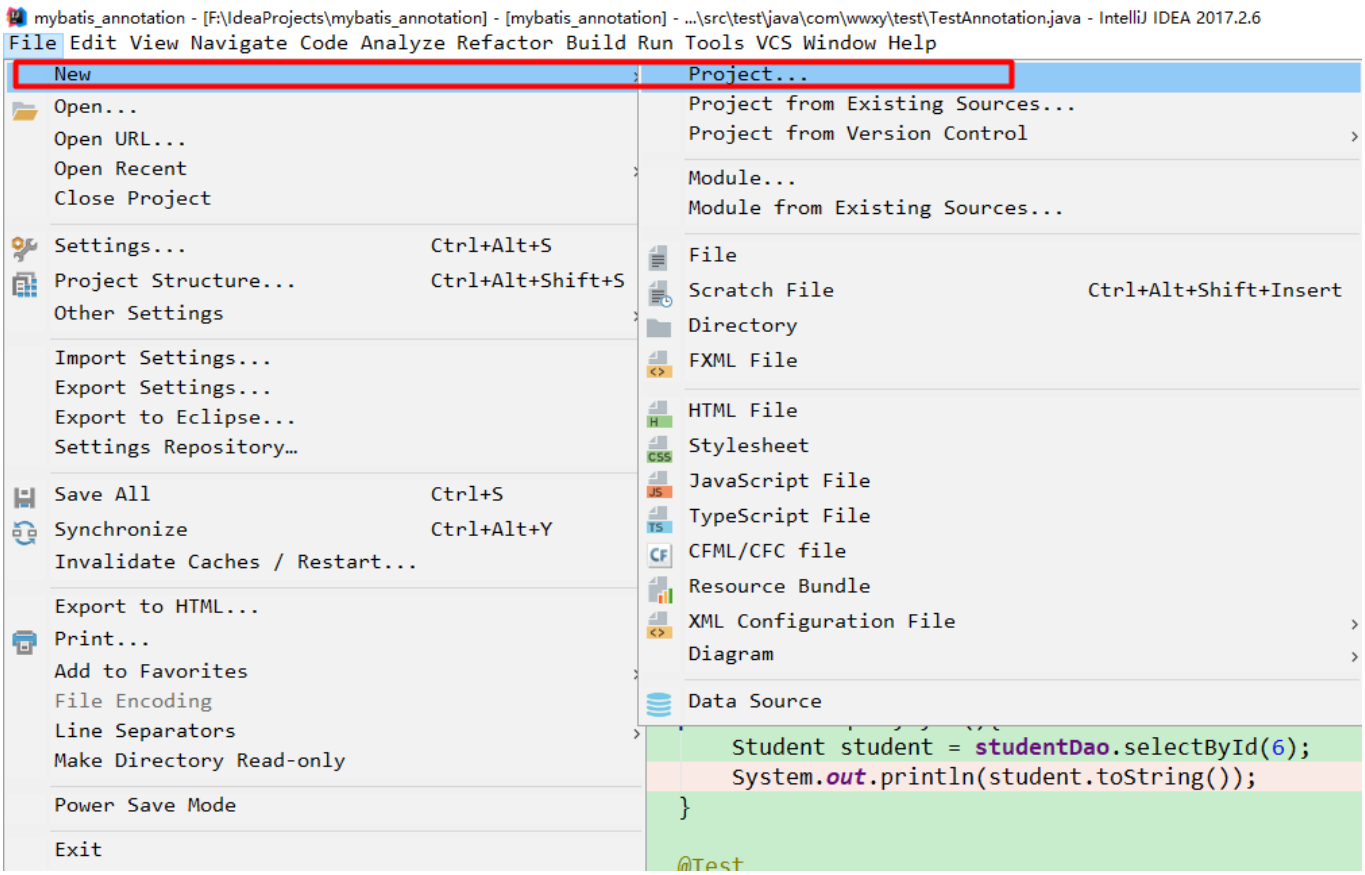
> 愿景
>
> 我们的愿景是成为 MyBatis 最好的搭档，就像 魂斗罗 中的 1P、2P，基友搭配，效率翻倍。

- **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

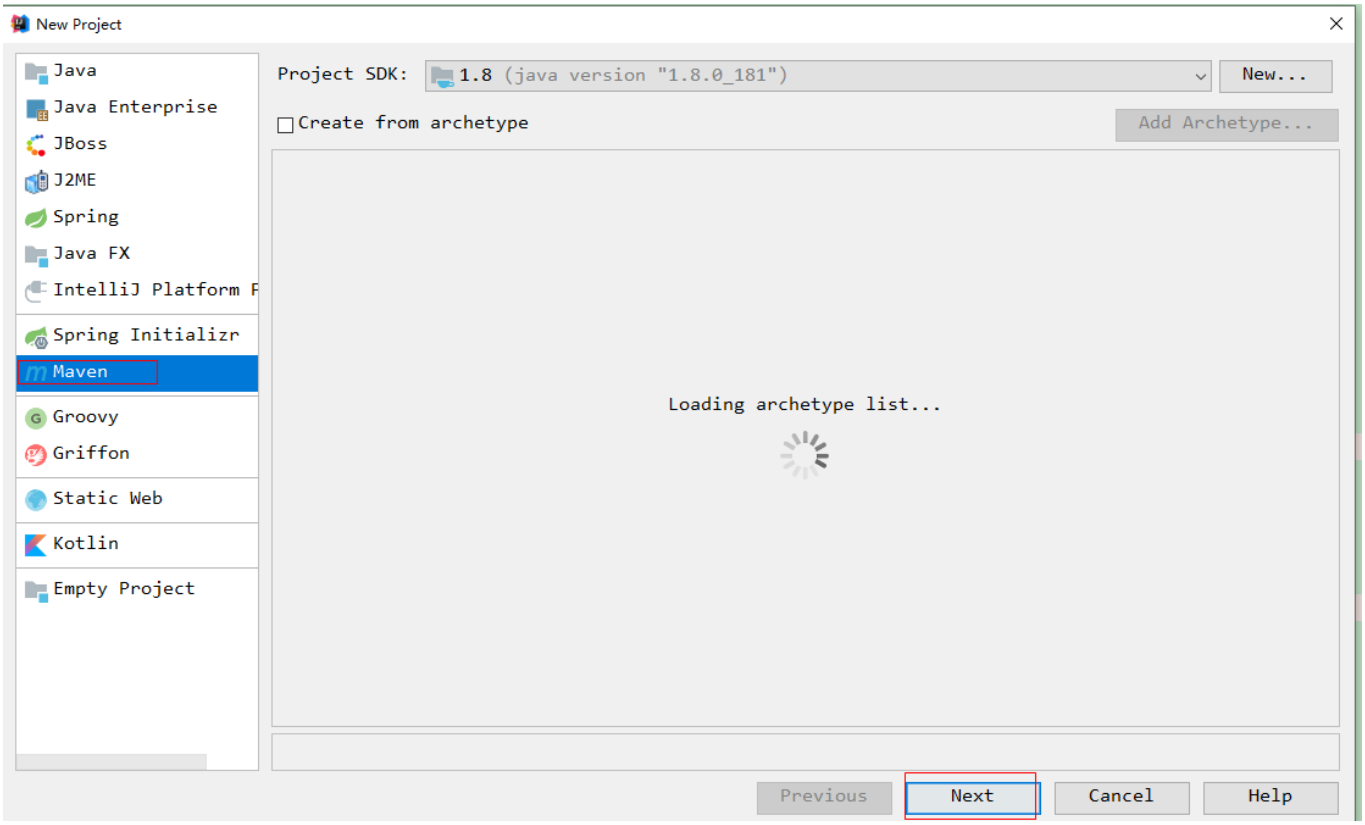
二、快速入门

2.1 环境搭建

2.1.1 创建Maven项目



选择Maven



2.1.2 导入pom坐标

注意：MyBatis及Mybatis-Spring依赖请勿加入项目配置，以免引起版本冲突，MyBatis-Plus会自动帮你维护。

```
<!-- mp 依赖 -->
<dependency>
     <groupId>com.baomidou</groupId>
     <artifactId>mybatis-plus</artifactId>
     <version>2.3</version>
</dependency>
<!--junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.9</version>
</dependency>
<!-- log4j -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<!-- c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.2</version>
</dependency>
<!-- mysql -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.37</version>
</dependency>
```

```
<!-- spring -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.10.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>4.3.10.RELEASE</version>
</dependency>
```

创建sqlMapConfig.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>
    <!--MyBatis的主配置文件，跟Spring整合后就不需要配置了-->
</configuration>
```

创建applicationContext.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.springframework.org/schema/tx"
        xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
        xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring-1.2.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
        <!-- 数据源 -->
<context:property-placeholder location="classpath:db.properties"/>
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${jdbc.driver}"></property>
    <property name="jdbcUrl" value="${jdbc.url}"></property>
    <property name="user" value="${jdbc.username}"></property>
    <property name="password" value="${jdbc.password}"></property>
</bean>
        <!-- 事务管理器 -->
<bean id="dataSourceTransactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>
        <!-- 基于注解的事务管理 -->
<tx:annotation-driven transaction-manager="dataSourceTransactionManager"/>
        <!-- 配置 SqlSessionFactoryBean -->
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
```

```
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>
</bean>
<!-- 配置 mybatis 扫描 mapper 接口的路径-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.mybatisPlus.dao"></property>
</bean>
</beans>
```

创建log4j.xml日志文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
        <param name="Encoding" value="UTF-8" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-5p %d{MM-dd
HH:mm:ss,SSS} %m (%F:%L) \n" />
        </layout>
    </appender>
    <logger name="java.sql">
        <level value="debug" />
    </logger>
    <logger name="org.apache.ibatis">
        <level value="info" />
    </logger>
    <root>
        <level value="debug" />
        <appender-ref ref="STDOUT" />
    </root>
</log4j:configuration>
```

创建db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/boot
jdbc.username=root
jdbc.password=root
```

测试在导入MyBatis-plus依赖的情况下，Mybatis是否能正常使用

Student

```
package com.mybatisPlus.pojo;

import java.io.Serializable;

public class Student implements Serializable{

    private int id;

    private String name;

    private int age;

    private String school;

    private int teacher_id;

    public int getId() {
        return id;
```

```
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSchool() {
        return school;
    }

    public void setSchool(String school) {
        this.school = school;
    }

    public int getTeacher_id() {
        return teacher_id;
    }

    public void setTeacher_id(int teacher_id) {
        this.teacher_id = teacher_id;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                '}';
    }
}
```

StudentDao

```
package com.mybatisPlus.dao;

import com.mybatisPlus.pojo.Student;
import org.apache.ibatis.annotations.Select;

import java.util.List;
```

```java
public interface StudentDao {

    @Select("select * from student")
    List<Student> findAll();

    Student selectById(Integer id);
}
```

StudentDao.xml

```xml
<?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.mybatisPlus.dao.StudentDao">

    <sql id="selectSql" >
        SELECT * from student
    </sql>

    <select id="selectById" parameterType="int" resultType="com.mybatisPlus.pojo.Student">
        <include refid="selectSql"></include> where id =#{id}
    </select>


</mapper>
```

Test

```java
import com.mybatisPlus.dao.StudentDao;
import com.mybatisPlus.pojo.Student;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.defaults.DefaultSqlSessionFactory;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;

public class MyBatisPlusTest {

    private ApplicationContext context = null;

    private SqlSession sqlSession;

    private DefaultSqlSessionFactory sqlSessionFactoryBean;

    @Before
    public void init(){
        if(context==null){
            context =  new ClassPathXmlApplicationContext("applicationContext.xml");
        }
```

```
        if(sqlSessionFactoryBean==null){
            //SqlSessionFactory是一个接口，这里需要找到其实现类
            sqlSessionFactoryBean = (DefaultSqlSessionFactory)context.getBean("sqlSessionFactoryBean");
        }
        if(sqlSession==null){
            sqlSession =  sqlSessionFactoryBean.openSession();
        }

    }



    @After
    public void destory(){
        sqlSession.commit();
        sqlSession.close();
    }






    @Test
    public void test1() throws Exception {
        StudentDao mapper = sqlSession.getMapper(StudentDao.class);
        Student student = mapper.selectById(1);
        System.out.println(student.toString());
    }


}
```

测试通过!

2.1.3 集成MP

集成MP只需要将applicationContext.xml中的org.mybatis.spring.SqlSessionFactoryBean替换为com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean,其他先不做修改就可以使用MyBatis-Plus

applicationContext.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.springframework.org/schema/tx"
        xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
        xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring-1.2.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
        <!-- 数据源 -->
<context:property-placeholder location="classpath:db.properties"/>
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${jdbc.driver}"></property>
    <property name="jdbcUrl" value="${jdbc.url}"></property>
```

```
    <property name="user" value="${jdbc.username}"></property>
    <property name="password" value="${jdbc.password}"></property>
</bean>
        <!-- 事务管理器 -->
<bean id="dataSourceTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>
        <!-- 基于注解的事务管理 -->
<tx:annotation-driven transaction-manager="dataSourceTransactionManager"/>
        <!-- 配置 SqlSessionFactoryBean -->
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>
</bean>
<!-- 配置 mybatis 扫描 mapper 接口的路径-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.mybatisPlus.dao"></property>
</bean>
</beans>
```

问题：SqlSessionFactoryBean替换成MybatisSqlSessionFactoryBean后还能使用Mybatis的注解开发和xml配置开发码？

答案是可以的，之前的查询依然可以使用，为什么呢？

可以对比MyBatis的SqlSessionFactoryBean和MyBatis-Plus的MybatisSqlSessionFactoryBean

```
public class SqlSessionFactoryBean implements FactoryBean<SqlSessionFactory>, InitializingBean,
ApplicationListener<ApplicationEvent> {
}


 public class MybatisSqlSessionFactoryBean implements FactoryBean<SqlSessionFactory>, InitializingBean,
ApplicationListener<ApplicationEvent>{
 }
```

在类的声明上，实现的接口都是一样的，所以我们认为SqlSessionFactoryBean能做的MybatisSqlSessionFactoryBean也可以做，这也符合MP只对Mybatis增强，而不对其修改的思想。

2.2.使用MP进行插入操作

创建StudentMapper接口

```
package com.mybatisPlus.dao;

import com.baomidou.mybatisplus.mapper.BaseMapper;
import com.mybatisPlus.pojo.Student;

/**
 * 注意：BaseMapper里的泛型是要操作的实体类的类型
 */
public interface StudentMapper extends BaseMapper<Student> {
}
```

创建测试类

Test

```
import com.mybatisPlus.dao.StudentMapper;
import com.mybatisPlus.pojo.Student;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestPlus {

    private ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

    private StudentMapper studentMapper = context.getBean("studentMapper",StudentMapper.class);



    @Test
    public void  test1(){
        Student st = new Student();
        st.setAge(20);
        st.setName("mybatisPlus");
        st.setSchool("wwxy");
        studentMapper.insert(st);  //测试成功
    }



}
```

2.2.1 @TableId注解

```
package com.baomidou.mybatisplus.annotations;

import com.baomidou.mybatisplus.enums.IdType;
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD})
public @interface TableId {
    String value() default "";

    IdType type() default IdType.NONE;
}
```

该注解有两个属性，value属性用于指定和该实体类id属性对应的表的id字段名称，若名称是一样的，该属性可以省略，

type属性用于指定该实体类对应表的主键策略。MP支持的主键策略如下：

> MP的主键策略

MP支持以下4中主键策略，可根据需求自行选用：

值	描述
IdType.AUTO	数据库ID自增
IdType.INPUT	用户输入ID
IdType.ID_WORKER	全局唯一ID，内容为空自动填充（默认配置）
IdType.UUID	全局唯一ID，内容为空自动填充

AUTO、INPUT和UUID大家都应该能够明白，这里主要讲一下ID_WORKER。首先得感谢开源项目 `Sequence` ，感谢作者 `李景枫` 。

什么是Sequence？简单来说就是一个分布式高效有序ID生产黑科技工具，思路主要是来源于 `Twitter-Snowflake算法` 。这里不详细讲解Sequence，有兴趣的朋友请 **点此去了解Sequence** 。

MP在Sequence的基础上进行部分优化，用于产生全局唯一ID，好的东西希望推广给大家，所以我们将ID_WORDER设置为默认配置。

一般Mysql数据库设置成IdType.AUTO，并需要在建表时，指定id的自增长。

2.2.2 @TableName注解

```
package com.baomidou.mybatisplus.annotations;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface TableName {
    //实体对应的表名
    String value() default "";
    //实体映射的结果集
    String resultMap() default "";
}
```

value属性用于指定与实体类对应的数据库表名称，当数据库名称和表名称不一致时需要设置该属性。默认是实体类名称的小写。

Student

```
package com.mybatisPlus.pojo;

import com.baomidou.mybatisplus.annotations.TableId;
import com.baomidou.mybatisplus.annotations.TableName;
import com.baomidou.mybatisplus.enums.IdType;

import java.io.Serializable;
@TableName(value = "student")
public class Student implements Serializable{

    @TableId(type = IdType.AUTO)
    private int id;
```

```
    private String name;

    private int age;

    private String school;

    private int teacher_id;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSchool() {
        return school;
    }

    public void setSchool(String school) {
        this.school = school;
    }

    public int getTeacher_id() {
        return teacher_id;
    }

    public void setTeacher_id(int teacher_id) {
        this.teacher_id = teacher_id;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                '}';
    }
}
```

2.2.3 @TableField注解

该注解用于指定在非主键属性的情况下和数据库字段的映射，

```
package com.baomidou.mybatisplus.annotations;

import com.baomidou.mybatisplus.enums.FieldFill;
import com.baomidou.mybatisplus.enums.FieldStrategy;
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD})
public @interface TableField {
    String value() default "";

    String el() default "";

    boolean exist() default true;

    String condition() default "%s=#{%s}";

    String update() default "";

    FieldStrategy strategy() default FieldStrategy.NOT_NULL;

    FieldFill fill() default FieldFill.DEFAULT;
}
```

该注解有两个重要的属性：

value：当实体类属性名称和数据库字段名称不一致时，需要设置该值为数据库与其对应的字段的名称

exist:指定该属性在数据库中是否有与其对应的字段，默认为true,若为true可以不配置该字段，若为false，则一定不需要配置，否则MP会把它也放到sql语句中，就会报错，即使是查询也会报错，这个和MyBatis有些不同。

2.2.4 使用MP插入如何获取插入数据的id值

MP对MyBatis框架进行了优化，不需要做任何操作就可以获取到刚插入表的id

2.3 MP的全局配置

在applicationContext.xml中配置bean

```
 <!-- 配置 SqlSessionFactoryBean -->
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>
```

```
    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
<bean id="globalConfiguration"  class="com.baomidou.mybatisplus.entity.GlobalConfiguration" >
    <!--指定实体类字段的下划线和数据库表字段驼峰命名的对应,2.3版本以后默认是true，所以也可以不指定-->
    <property name="dbColumnUnderline" value="true"></property>

    <!--指定全局主键策略，不需要再一个类一个类的配置-->
    <property name="idType" value="0"></property>

    <!--全局表前缀配置-->
    <property name="tablePrefix" value="tbl_"></property>

</bean>
```

2.3.1 配置下划线和驼峰名称字段的自动映射

配置了这个，就可以实现带有下划线的属性名称和数据库字段的驼峰命名的字段的映射了，反着命名也是可以的

比如：

实体类中： teacherId;

表中：teacher_id;

或者：

实体类中：teacher_id;

表中：teacherId

都是不需要再单独配置映射的，可以自动映射

2.3.1.1 驼峰命名法

骆驼式命名法就是当变量名或函数名是由一个或多个单词连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；从第二个单词开始以后的每个单词的首字母都采用大写字母，例如：myFirstName、myLastName，这样的变量名看上去就像骆驼峰一样此起彼伏，故得名。

- 小驼峰法

  变量一般用小驼峰法标识。驼峰法的意思是：除第一个单词之外，其他单词首字母大写，通常用于变量的命名

  int studentCount;

- 大驼峰法

  每一个单词的首字母都要大写，通常用于类名的定义

  StudentService.java

MP的变量映射既支持小驼峰命名，又支持大驼峰命名。

```
<!--指定实体类字段的下划线和数据库表字段驼峰命名的对应,2.3版本以后默认是true，所以也可以不指定-->
    <property name="dbColumnUnderline" value="true"></property>
```

2.3.2 指定全局主键策略

```
<!--指定全局主键策略，不需要再一个类一个类的配置-->
    <property name="idType" value="0"></property>
```

将idType的值配置成0，是什么意思呢？追踪GlobalConfiguration的源码可以看到

```
public class GlobalConfiguration implements Serializable {
    ...
    private IdType idType;
    ...
    public void setIdType(int idType) {
        this.idType = IdType.getIdType(idType);
    }
}
```

查看IdType的代码

```
package com.baomidou.mybatisplus.enums;

public enum IdType {
    AUTO(0, "数据库ID自增"),
    INPUT(1, "用户输入ID"),
    ID_WORKER(2, "全局唯一ID"),
    UUID(3, "全局唯一ID"),
    NONE(4, "该类型为未设置主键类型"),
    ID_WORKER_STR(5, "字符串全局唯一ID");

    private final int key;
    private final String desc;

    private IdType(int key, String desc) {
        this.key = key;
        this.desc = desc;
    }

    public static IdType getIdType(int idType) {
        IdType[] its = values();
        IdType[] arr$ = its;
        int len$ = its.length;

        for(int i$ = 0; i$ < len$; ++i$) {
            IdType it = arr$[i$];
            if (it.getKey() == idType) {
                return it;
            }
        }

        return ID_WORKER;
    }

    public int getKey() {
        return this.key;
    }

    public String getDesc() {
        return this.desc;
    }
}
```

可以看出，当设置成0时，主键策略是AUTO,也就是主键自增

2.3.3 指定表前缀

一般情况下，MP是默认以实体类的小写来作为表的名称的，如Student类对应的表名就是student，但如果定义的表的名称和实体类不对应，我们可以使用@TableName（value=""）来指定数据库的表名称。

如果我们对数据库表的名称命名有规律，可以通过添加前缀的方式进行全局的表名称配置。

例如：

tbl_student ---> Student

tbl_teacher ---> Teacher

```
<!--全局表前缀配置-->
    <property name="tablePrefix" value="tbl_"></property>
```

2.4 MP的选择性插入和全字段插入

insert(T t)这个是选择性插入，当对象中某个字段没有值，就不会将其拼接到sql语句中，

insertAllColumn(T t)：全字段插入，不管对象有么有设置值，都会将全字段拼接到sql中，没有值为null

2.5 更新操作

updateById(T t):根据id局部修改实体类，没有设置值得属性不会写到sql中，即只有在实体类中配置的值的字段才会被修改。除了id

updateAllColumnById(T t): 根据id进行全字段的修改，没有设置值的字段为null

> 注意：
>
> 如果我们在实体类中对于整型的变量定义为了 int 那它是有默认值的，那就是0，所以在调用updateById时也就会将对应的字段修改为0，所以我们以后再定义实体类时尽量使用其包装类。

2.6 查询操作

> T selectById(Serializable var1):根据id查询数据，传入的参数可以是实现了Serializable的实体类，将id设置进去，也可以直接传int类型的id值
>
> T selectOne(T t):根据条件查询数据，查询条件封装到实体类中，返回的结果只有一条数据，如果返回多条记录，该方法会报错。所以该方法比较鸡肋，该方法执行的sql如下：
>
> ```
> SELECT id AS id,`name` AS userName,age,school,teacher_id AS TeacherId FROM student WHERE id=?
> ```

> List selectBatchIds(List list):根据id集合进行查询，该方法执行的sql如下：
>
> ```
> SELECT id AS id,`name` AS userName,age,school,teacher_id AS TeacherId FROM student WHERE id IN ( ? , ? , ? )
> ```
>
> List selectByMap(Map<String, Object> columnMap): 根据map集合里的数据作为查询条件，columnMap里的键是数据库里的字段的名称。该方法执行的sql如下：
>
> ```
> SELECT id AS id,`name` AS userName,age,school,teacher_id AS TeacherId FROM student WHERE name = ? AND age = ?
> ```
>
> 注意：如果map里的某个键值对的值为null，则不会将其作为查询条件。
>
> List selectPage(RowBounds rowBounds, @Param("ew") Wrapper wrapper):分页查询，传入RowBounds对象作为查询条件，Wrapper对象是条件构造器，可以先传null。该分页使用的是Mybstis的分页，并不是物理分页。要实现真正的物理分页MP有自己的分页插件。该方法执行的sql如下：
>
> ```
> SELECT id AS id,`name` AS userName,age,school,teacher_id AS TeacherId FROM student
> ```

2.7 删除操作

> Integer deleteById(Serializable id)：根据id删除
>
> Integer deleteByMap(@Param("cm") Map<String, Object> columnMap)：根据参数删除
>
> Integer deleteBatchIds(List<? extends Serializable> idList)：根据id集合删除

2.8 MP启动注入SQL原理分析

问题：

创建xxxMapper接口继承了BaseMapper后，使用的增删改查的方法都来自于BaseMapper，那sql语句是在哪里写的呢？

三、条件构造器 EntityWrapper

3.1 EntityWrapper简介

3.3.1 定义

MyBatis-plus通过EntityWrapper（简称 EW，MP 封装的一个查询条件构造器）或者Condition来让用户自由的构建查询条件，简单便捷，没有额外的负担，能够提高开发效率

3.3.2 作用

它是一个实体包装器，主要用于处理sql拼接，排序，实体参数查询等

注意：当通过EW设置参数时，指定的参数名称必须是数据库的字段名称，而不是实体类的属性名称。

3.4 常用方法

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

注意：使用last方法是直接在sql语句最后进行拼接，有sql注入的风险。

3.3.4 测试几个常用的方法

1. 分页查询student表中名字姓张的且学校是清华的学生

```
@Test
   public void test6() {
    List<Student> students = studentMapper.selectPage(new Page<Student>(2, 2), new EntityWrapper<Student>
().like("name", "张").eq("school", "清华"));
        System.out.println(students);
   }
```

2. 查询student表中姓张且年龄小于30岁的的学生

```
@Test
   public void test7() {
   List<Student> students = studentMapper.selectList(new EntityWrapper<Student>
().like("name","张").lt("age",30));
        System.out.println(students);
    }
```

3. 将student表中姓名为张三丰的年龄是100的用户的学校修改为武当

```
@Test
    public void test8() {
        Student student = new Student();
        student.setSchool("武当");
        Integer update = studentMapper.update(student, new EntityWrapper<Student>().eq("name", "张三
丰").eq("age", "100"));

    }
```

4.分页查询student表中名字姓张的且学校是清华的学生,条件构造器使用的是Condititon

```
@Test
    public void test6() {
        List<Student> students = studentMapper.selectPage(new Page<Student>(1, 2),
Condition.create().like("name", "张").eq("school", "清华"));
        System.out.println(students);
    }
```

Condition和EntityWrapper只有获取对象的方式不同，其他用法基本一样。

四、ActiveRecord(活动记录)

4.1 定义

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。

Active Record一直广受动态语言（PHP,Buby等）的喜爱，而Java作为准静态语言，对于Active Record往往只能感叹其优雅，所以MP也在AR的道路上进行了一定的探索。

4.2 如何使用AR模式

只需让实体类继承Model 类并实现主键指定方法，即可开启AR之旅

创建一个Teacher表和一个Teacher实体类

```
CREATE TABLE `teacher` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `tName` varchar(10) COLLATE utf8_bin DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `className` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

```
package com.mybatisPlus.pojo;

import com.baomidou.mybatisplus.activerecord.Model;
import com.baomidou.mybatisplus.annotations.TableName;

import java.io.Serializable;
public class Teacher extends Model<Teacher>{

    private Integer id;

    private String tName;

    private Integer age;

    private String className;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String gettName() {
        return tName;
    }

    public void settName(String tName) {
        this.tName = tName;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getClassName() {
        return className;
    }

    public void setClassName(String className) {
        this.className = className;
    }

    @Override
    public String toString() {
```

```
        return "Teacher{" +
                "id=" + id +
                ", tName='" + tName + '\'' +
                ", age=" + age +
                ", className='" + className + '\'' +
                '}';
    }

    //指定该实体类中哪个属性和数据库的id字段对应
    @Override
    protected Serializable pkVal() {
        return id;
    }
}
```

Test

```
@Test
    public void test9(){
        Teacher t = new Teacher();
        Teacher teacher = t.selectById(1);
        System.out.println(teacher);
    }
```

执行发现报错：

```
java.lang.NullPointerException
    at com.baomidou.mybatisplus.toolkit.GlobalConfigUtils.currentSessionFactory(GlobalConfigUtils.java:60)
    at com.baomidou.mybatisplus.mapper.SqlHelper.sqlSession(SqlHelper.java:104)
    at com.baomidou.mybatisplus.mapper.SqlHelper.sqlSession(SqlHelper.java:58)
    at com.baomidou.mybatisplus.activerecord.Model.sqlSession(Model.java:373)
    at com.baomidou.mybatisplus.activerecord.Model.selectById(Model.java:228)
    at TestPlus.test9(TestPlus.java:104)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:44)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:41)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:69)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:48)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:292)
    at org.junit.runner.JUnitCore.run(JUnitCore.java:157)
    at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:68)
    at com.intellij.rt.execution.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:47)
    at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:242)
    at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:70)
```

后来在网上查资料才知道，要使用AR模式。还必须创建一个接口并实现BaseMapper接口，里面的方法才开始使用。话说我接口都创建了还这么麻烦实现AR干啥？

TeacherMapper

```
package com.mybatisPlus.dao;

import com.baomidou.mybatisplus.mapper.BaseMapper;
import com.mybatisPlus.pojo.Teacher;

public interface TeacherMapper  extends BaseMapper<Teacher> {
}
```

重新测试通过!

4.3 测试AR模式的其他方法

```
@Test
   public void test9(){
       Teacher t = new Teacher();
       t.setAge(20);
       t.setClassName("清华班");
       t.settName("毕向东");
       t.insert();
   }
```

测试时报了一个错:

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 't_name' in 'field list'
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.Util.getInstance(Util.java:387)
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:941)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3870)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3806)
    at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2470)
    at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2617)
    at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2550)
    at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1861)
    at com.mysql.jdbc.PreparedStatement.execute(PreparedStatement.java:1192)
    at com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.execute(NewProxyPreparedStatement.java:67)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.apache.ibatis.logging.jdbc.PreparedStatementLogger.invoke(PreparedStatementLogger.java:59)
    at com.sun.proxy.$Proxy18.execute(Unknown Source)
    at org.apache.ibatis.executor.statement.PreparedStatementHandler.query(PreparedStatementHandler.java:63)
    at org.apache.ibatis.executor.statement.RoutingStatementHandler.query(RoutingStatementHandler.java:79)
    at org.apache.ibatis.executor.SimpleExecutor.doQuery(SimpleExecutor.java:63)
    at org.apache.ibatis.executor.BaseExecutor.queryFromDatabase(BaseExecutor.java:326)
    at org.apache.ibatis.executor.BaseExecutor.query(BaseExecutor.java:156)
    at org.apache.ibatis.executor.CachingExecutor.query(CachingExecutor.java:109)
    at org.apache.ibatis.executor.CachingExecutor.query(CachingExecutor.java:83)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:148)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:141)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectOne(DefaultSqlSession.java:77)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
```

```
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at
com.baomidou.mybatisplus.MybatisSqlSessionTemplate$SqlSessionInterceptor.invoke(MybatisSqlSessionTemplate.java:401)

    at com.sun.proxy.$Proxy10.selectOne(Unknown Source)
    at com.baomidou.mybatisplus.MybatisSqlSessionTemplate.selectOne(MybatisSqlSessionTemplate.java:131)
    at com.baomidou.mybatisplus.activerecord.Model.selectById(Model.java:228)
    at com.baomidou.mybatisplus.activerecord.Model.selectById(Model.java:242)
    at TestPlus.test10(TestPlus.java:115)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:44)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:41)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:69)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:48)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:292)
    at org.junit.runner.JUnitCore.run(JUnitCore.java:157)
    at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:68)
    at com.intellij.rt.execution.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:47)
    at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:242)
    at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:70)
DEBUG 04-05 19:19:09,575 Attempted to convert SQLException to SQLException. Leaving it alone. [SQLState: 42S22;
errorCode: 1054] (SqlUtils.java:97)
```

意思就是数据库中没有叫“t_name”的字段，我原本以为实体类的属性名称和数据库字段名称一致会优先于全局的驼峰名称和下划线映射的配置。现在看来不是的。通过使用@TableField注解指定或者将全局驼峰映射关闭就可以正常插入。这个还是有些遗憾的。

4.4 AR模式的其他方法

1）插入操作 public boolean insert() 2) 修改操作 public boolean updateById() 3) 查询操作 public T selectById() public T selectById(Serializable id) public List selectAll() public List selectList(Wrapper wrapper) public int selectCount(Wrapper wrapper)

4）删除操作

public boolean deleteById()

public boolean deleteById(Serializable id)

public boolean delete(Wrapper wrapper)

5) 分页复杂操作

public Page selectPage(Page page, Wrapper wrapper)

五、代码生成器

5.1 代码生成器的功能

MP提供了大量的自定义设置，生成的代码完全满足各类型的需求，

5.2 MP的代码生成器和MyBatis的MBG代码生成器比较

- MP的代码生成器都是基于Java代码生成，MBG基于xml文件生成。
- MyBatis的代码生成器可以生成实体类，Mapper接口，Mapper映射文件。MP的代码生成器可以生成实体类、Mapper接口，Mapper映射文件，Service层，Controller层。

5.3 使用代码生成器

创建代码生成器方法

```
public static void createCode(){
        //全局配置
        GlobalConfig config = new GlobalConfig();
        config  .setActiveRecord(true)//是否支持AR模式
                .setAuthor("孔")//设置作者
                .setOutputDir("F:/IdeaProjects/mybatis_plus/src/main/java")  //文件输出路径
                .setFileOverride(true)//是否文件覆盖
                .setBaseColumnList(true) //xml 生成xml映射文件里的基本的列
                .setBaseResultMap(true)//xml  生成字段映射<resultMap>
                .setEnableCache(false) //是否启用二级缓存
                .setServiceName("%sService")//设置生成的service接口名字首字母是否带I
                .setControllerName("%sController")
                .setIdType(IdType.AUTO);  //设置主键自增
        //数据源配置
        DataSourceConfig sourceConfig = new DataSourceConfig();
        sourceConfig.setDbType(DbType.MYSQL)  //设置数据库类型
                .setUrl("jdbc:mysql://localhost:3306/boot")
                .setDriverName("com.mysql.jdbc.Driver")
                .setUsername("root")
                .setPassword("root");

        //策略配置
        StrategyConfig strategyConfig = new StrategyConfig();
        strategyConfig.setNaming(NamingStrategy.underline_to_camel) //数据库到映射实体的命名策略
        .setDbColumnUnderline(false) //是否开启表名和字段名使用下划线命名
                .setCapitalMode(true); //是否开启全局大写

        // 包名策略
        PackageConfig packageConfig = new PackageConfig();
        packageConfig.setParent("com.mybatisPlus")  //设置父包
        .setController("controller")
        .setEntity("pojo")
        .setMapper("dao")
        .setService("service")
        .setServiceImpl("service")
        .setXml("dao");

        AutoGenerator autoGenerator = new AutoGenerator();

autoGenerator.setGlobalConfig(config).setDataSource(sourceConfig).setPackageInfo(packageConfig).setStrategy(strategyConfig);

        autoGenerator.execute();
    }
```

执行后就可以将代码生成到指定的目录

在执行测试时出现了一个问题，当xml映射文件和mapper接口在同一个包下时，运行项目会报错，意思是找不到xml映射文件，

```
org.apache.ibatis.binding.BindingException: Invalid bound statement (not found)
```

解决:

在pom添加如下坐标就可以了

```
<build>
        <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
        </resources>
 </build>
```

意思是将src/main/java下也指定为resources目录

六、MP的插件扩展

6.1 MyBatis的插件机制

6.1.1 插件机制:

Mybatis 通过插件(Interceptor) 可以做到拦截四大对象相关方法的执行,根据需求，完

成相关数据的动态改变。

Executor

StatementHandler

ParameterHandler

ResultSetHandler

6.1.2 插件原理

四大对象的每个对象在创建时，都会执行 interceptorChain.pluginAll()，会经过每个插

件对象的 plugin()方法，目的是为当前的四大对象创建代理。代理对象就可以拦截到四

大对象相关方法的执行，因为要执行四大对象的方法需要经过代理

6.2 分页插件

之前说过，MP自带的selectPage的分页方法使用得当MyBatis底层的RowBounds对象进行的内存分页，是将所有的数据查出来通过代码进行的分页，所以MP又提供了一个物理分页的插件，即com.baomidou.mybatisplus.plugins.PaginationInterceptor

使用插件不需要进行依赖导入，只需要在MyBatis的配置文件中注册一下即可，由于MyBatis和Spring整合后，配置文件不再使用，所以也可以在Spring的配置文件中使用。

- 在MyBatis的sqlMapConfig.xml中配置

```
<?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>
<!--MyBatis的主配置文件，跟Spring整合后就不需要配置了-->
   <plugins>
     <plugin interceptor="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></plugin>
    </plugins>
</configuration>
```

或者在Spring的applicationContext中配置。应该配置在SqlSessionFactoryBean的属性中，当然这里是MP提供的MyBatisSqlSessionFactoryBean

```
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>

    <property name="plugins">
        <list>
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></bean>
        </list>
    </property>

    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
```

6.3 执行分析插件

1）使用对象：com.baomidou.mybatisplus.plugins.SqlExplainInterceptor

2) SQL 执行分析拦截器，只支持 MySQL5.6.3 以上版本

3) 该插件的作用是分析 DELETE UPDATE 语句,防止小白

或者恶意进行 DELETE UPDATE 全表操作

4) 只建议在开发环境中使用，不建议在生产环境使用

5) 在插件的底层 通过 SQL 语句分析命令:Explain 分析当前的 SQL 语句，

根据结果集中的 Extra 列来断定当前是否全表操作。

配置插件

```
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>

    <property name="plugins">
        <list>
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></bean>
```

```
            <bean class="com.baomidou.mybatisplus.plugins.SqlExplainInterceptor">
                <!--如果判断是全表删除，就抛出异常-->
                <property name="stopProceed" value="true"></property>
            </bean>
        </list>
    </property>

    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
```

Test

```
@Test
   public void test2(){
      Student student = new Student();
      student.delete(null);
  }
```

进行全表删除，报了异常

```
Caused by: org.apache.ibatis.exceptions.PersistenceException:
Cause: com.baomidou.mybatisplus.exceptions.MybatisPlusException:
com.baomidou.mybatisplus.exceptions.MybatisPlusException: Error: Full table operation is prohibited. SQL:
DELETE FROM student
```

6.4 性能分析插件

1) com.baomidou.mybatisplus.plugins.PerformanceInterceptor

2) 性能分析拦截器，用于输出每条 SQL 语句及其执行时间

3) SQL 性能执行分析,开发环境使用， 超过指定时间，停止运行。有助于发现问题

配置性能分析插件

```
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>

    <property name="plugins">
        <list>
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></bean>

            <!--执行分析插件-->
            <bean class="com.baomidou.mybatisplus.plugins.SqlExplainInterceptor">
                <!--如果判断是全表删除，就抛出异常-->
                <property name="stopProceed" value="true"></property>
            </bean>

                <!--性能分析插件-->
                <bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor">
                    <!--是否格式化sql-->
                    <property name="format" value="true"></property>
                    <!--配置sql的执行最长时间。毫秒值-->
```

```
                    <property name="maxTime" value="10000"></property>
                </bean>

        </list>
    </property>

    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
```

aplicationContext.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.springframework.org/schema/tx"
        xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
        xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring-1.2.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
        <!-- 数据源 -->
<context:property-placeholder location="classpath:db.properties"/>
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${jdbc.driver}"></property>
    <property name="jdbcUrl" value="${jdbc.url}"></property>
    <property name="user" value="${jdbc.username}"></property>
    <property name="password" value="${jdbc.password}"></property>
</bean>
        <!-- 事务管理器 -->
<bean id="dataSourceTransactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>
        <!-- 基于注解的事务管理 -->
<tx:annotation-driven transaction-manager="dataSourceTransactionManager"/>
        <!-- 配置 SqlSessionFactoryBean -->
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>

    <property name="plugins">
        <list>
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></bean>

            <!--执行分析插件-->
            <bean class="com.baomidou.mybatisplus.plugins.SqlExplainInterceptor">
                <!--如果判断是全表删除，就抛出异常-->
                <property name="stopProceed" value="true"></property>
            </bean>
```

```
            <!--性能分析插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor">
                <!--是否格式化sql-->
                <property name="format" value="true"></property>
                <!--配置sql的执行最长时间。毫秒值-->
                <property name="maxTime" value="10000"></property>
            </bean>

        </list>
    </property>

    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
<!-- 配置 mybatis 扫描 mapper 接口的路径-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.mybatisPlus.dao"></property>
</bean>

<bean id="globalConfiguration"  class="com.baomidou.mybatisplus.entity.GlobalConfiguration" >
    <!--指定实体类字段的下划线和数据库表字段驼峰命名的对应,2.3版本以后默认是true，所以也可以不指定-->
    <property name="dbColumnUnderline" value="false"></property>

    <!--指定全局主键策略，不需要再一个类一个类的配置-->
    <property name="idType" value="0"></property>

    <!--全局表前缀配置-->
    <!--<property name="tablePrefix" value="tbl_"></property>-->

</bean>
```

Test

```java
 @Test
    public void test() {
        Teacher teacher = new Teacher();
        Page<Teacher> page = new Page<Teacher>(2, 2);
        Page<Teacher> teacherPage = teacher.selectPage(page, null);
        List<Teacher> records = teacherPage.getRecords();
        System.out.println(records);

        System.out.println("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());

  }
```

执行后的控制台输出

```
Time: 8 ms - ID: com.mybatisPlus.dao.TeacherMapper.selectPage
 Execute SQL:
    SELECT
        id AS id,
```

```
        tName,
        age,
        className
    FROM
        teacher LIMIT 2,
        2]

[Teacher{, id=3, tName=毕向东, age=20, className=清华班}, Teacher{, id=4, tName=毕向东, age=20, className=清华班}]
总条数：5
当前页码：2
总页码：3
每页显示条数：2
是否有上一页：true
是否有下一页：true
```

会发现有这个方法执行的sql和执行的时间，如果超过了之前设置的最长执行时间，则会报错，显示某条sql执行的时间过长。

6.5 乐观锁插件

1) com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor

2) 如果想实现如下需求: 当要更新一条记录的时候，希望这条记录没有被别人更新

3) 乐观锁的实现原理:

取出记录时，获取当前 version 2

更新时，带上这个 version 2

执行更新时， set version = yourVersion+1 where version = yourVersion

如果 version 不对，就更新失败

4) @Version 用于注解实体字段，必须要有。

配置乐观锁

```
<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:SqlMapConfig.xml"></property>
    <!-- 别名处理 -->
    <property name="typeAliasesPackage" value="com.mybatisPlus.pojo"></property>

    <property name="plugins">
        <list>
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></bean>

            <!--执行分析插件-->
            <bean class="com.baomidou.mybatisplus.plugins.SqlExplainInterceptor">
                <!--如果判断是全表删除，就抛出异常-->
                <property name="stopProceed" value="true"></property>
            </bean>

            <!--性能分析插件-->
            <bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor">
                <!--是否格式化sql-->
                <property name="format" value="true"></property>
                <!--配置sql的执行最长时间。毫秒值-->
                <property name="maxTime" value="10000"></property>
            </bean>
```

```
            <!--注册分页插件-->
            <bean class="com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor"></bean>
        </list>
    </property>

    <!--引入全局配置-->
    <property name="globalConfig" ref="globalConfiguration"></property>
</bean>
```

七、自定义全局操作

根据 MybatisPlus 的 AutoSqlInjector 可以自定义各种你想要的 sql ,注入到全局中，相当于自 定义 Mybatisplus 自动注入的方法。之前需要在 xml 中进行配置的 SQL 语句，现在通过扩展 AutoSqlInjector 在加载 mybatis 环境时就注入。

具体操作步骤

- 在Mapper接口中自定义方法

```
package com.mybatisPlus.dao;

import com.mybatisPlus.pojo.People;
import com.baomidou.mybatisplus.mapper.BaseMapper;

import java.util.List;

/**
 * <p>
 *  Mapper 接口
 * </p>
 *
 * @author 孔
 * @since 2020-04-06
 */
public interface PeopleMapper extends BaseMapper<People> {

    List<People> findPeopleByName();
}
```

- 创建自定义注入器对象并继承AutoSqlInjector 类，重写Inject方法

```
package com.mybatisPlus.injector;

import com.baomidou.mybatisplus.entity.TableInfo;
import com.baomidou.mybatisplus.mapper.AutoSqlInjector;
import org.apache.ibatis.builder.MapperBuilderAssistant;
import org.apache.ibatis.mapping.SqlSource;
import org.apache.ibatis.session.Configuration;

/**
 * 全局自定义操作
 */
public class MysqlInjector  extends AutoSqlInjector {

    @Override
    public void inject(Configuration configuration, MapperBuilderAssistant builderAssistant, Class<?>
```

```
mapperClass, Class<?> modelClass, TableInfo table) {
        //将mapper接口中定义的方法处理成对应的MappedStatement对象，加入到congifuration对象中
        String sql= "select * from people";
        //构造sqlSource对象
        SqlSource sqlSource = this.languageDriver.createSqlSource(this.configuration, sql, modelClass);
        //定义方法名称
        String funName = "findPeopleByName";
        this.addSelectMappedStatement(mapperClass, funName, sqlSource, modelClass, table);
    }
}
```

注意：在最后一行addSelectMappedStatement()的调用需要看具体情况，因为我定义的时查询的方法，所以调用的时addSelect***，如果是其他方法，应该调用其他的方法，否则会启动失败

- 将自定义注入器交给Spring容器管理对象，并同时在全局配置对象里注册自定义注入器

```
<bean id="globalConfiguration"  class="com.baomidou.mybatisplus.entity.GlobalConfiguration" >
    <!--指定实体类字段的下划线和数据库表字段驼峰命名的对应,2.3版本以后默认是true，所以也可以不指定-->
    <property name="dbColumnUnderline" value="false"></property>

    <!--指定全局主键策略，不需要再一个类一个类的配置-->
    <property name="idType" value="0"></property>

    <!--全局表前缀配置-->
    <!--<property name="tablePrefix" value="tbl_"></property>-->

    <!--注册全局操作注入器-->
    <property name="sqlInjector" ref="mysqlInjector"></property>
</bean>

<!--自定义全局操作注入器-->
    <bean id="mysqlInjector" class="com.mybatisPlus.injector.MysqlInjector"></bean>
```

- 测试

```
@Test
    public void test2() {
        PeopleMapper peopleMapper = context.getBean("peopleMapper", PeopleMapper.class);
        List<People> peopleByName = peopleMapper.findPeopleByName();
        System.out.println(peopleByName);


    }
```

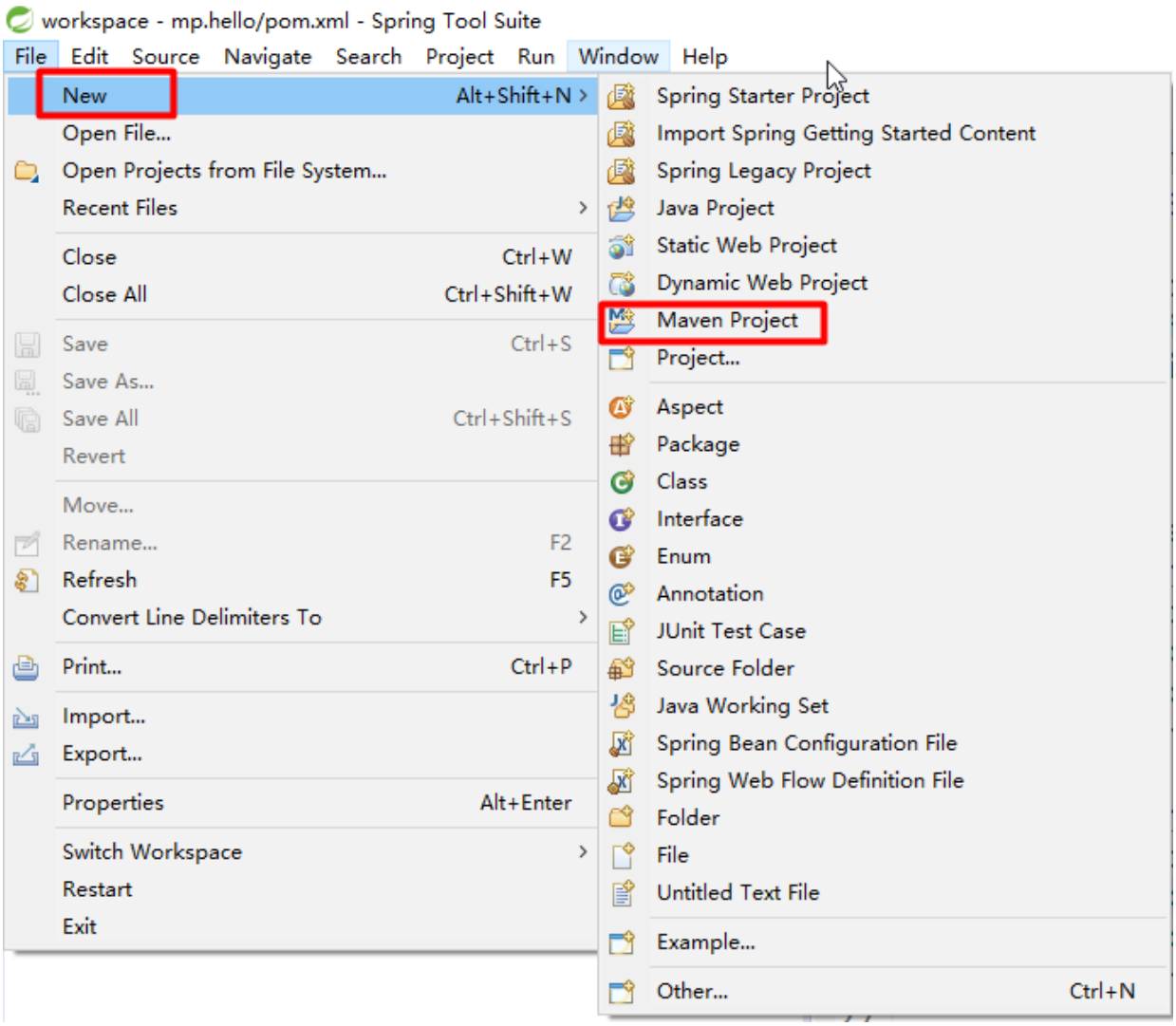
成功！，注意注入器指定定义一个，所以这个插件个人感觉不太实用。

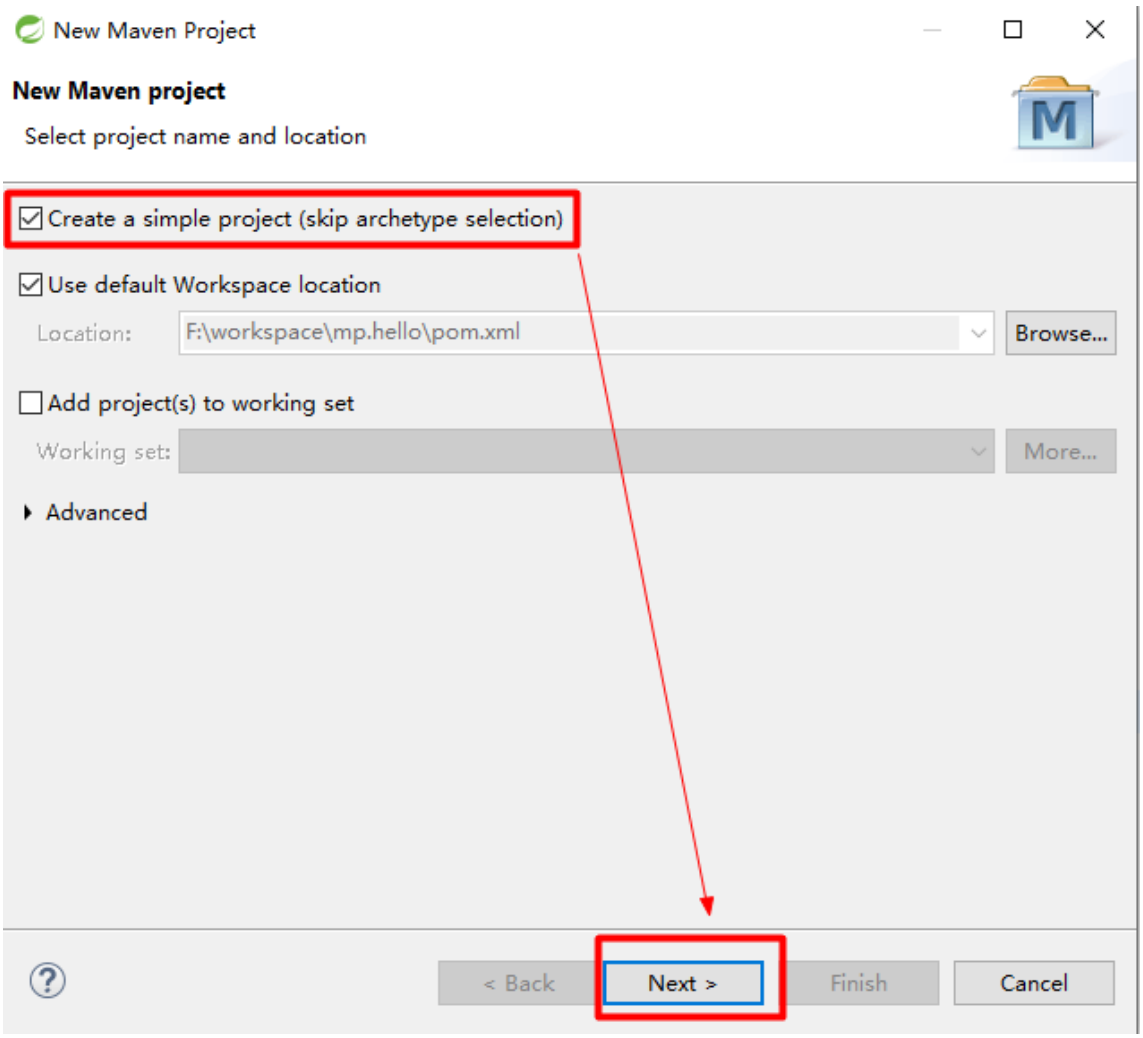
八 Oracle 主键 Sequence

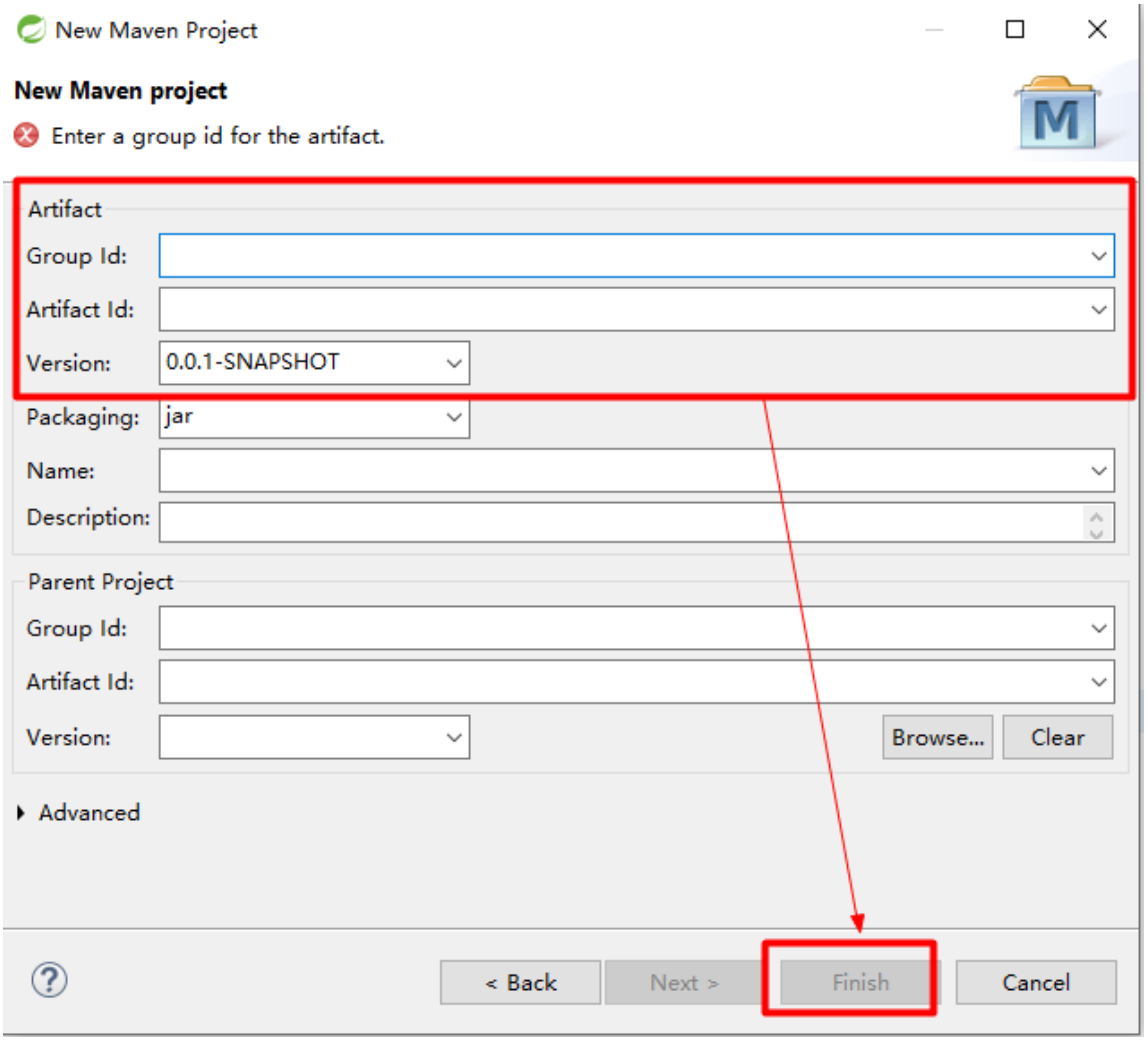
后续补充

九、SpringBoot2.x集成MyBatis-Plus3

9.1 创建Maven项目


workspace - mp.hello/pom.xml - Spring Tool Suite
File
Edit
Source
Navigate
Search
Project
Run
Window
Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Spring Starter Project
Import Spring Getting Started Content
Spring Legacy Project
Java Project
Static Web Project
Dynamic Web Project
Maven Project
Project...
Aspect
Package
Class
Interface
Enum
Annotation
JUnit Test Case
Source Folder
Java Working Set
Spring Bean Configuration File
Spring Web Flow Definition File
Folder
File
Untitled Text File
Example...
Other... Ctrl+N


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
F:\workspace\mp.hello\pom.xml
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel



9.2 导入pom坐标

```
<!-- SpringBoot的父工程 -->
  <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.3.RELEASE</version>
  </parent>


  <dependencies>
    <!-- springboot的启动器 -->
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <!-- springweb -->
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- spring-test -->
    <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
    </dependency>

    <!-- lombok简化代码，省去写实体类的get/set方法 -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.12</version>
        <optional>true</optional>
    </dependency>

    <!-- MyBatis-Plus的starter -->
     <dependency>
             <groupId>com.baomidou</groupId>
             <artifactId>mybatis-plus-boot-starter</artifactId>
             <version>3.1.0</version>
    </dependency>

   <!-- mysql驱动，版本已经在springboot的父工程里定义了 -->
  <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
   </dependency>

   <!-- 代码生成器 -->
   <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.1.0</version>
    </dependency>

   <!-- 代码生成器模板 -->
   <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
    </dependency>

   <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
   </dependency>
  </dependencies>


  <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

9.3 application.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/boot?autoReconnect=true&useUnicode=true&characterEncoding=utf-8
    username: root
    password: root
mybatis-plus:
    # 指定mapper的xml文件的路径
    mapper-locations: classpath*:mapper/*.xml
    #实体扫描，指定这个就可以使用实体类名称的小写来代替全限定类名了
    typeAliasesPackage: mp.demo.pojo
    global-config:
      id-type: auto #设置主键自增，适用于mysql
      column-underline: false #驼峰下划线转换关闭
```

9.4 启动类

```
package mp.demo;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@MapperScan("mp.demo.mapper")
public class Start {

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

}
```

9.5 MP的插件配置

```
package mp.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor;

@Configuration
public class MyBatisPlusConfig {

    /**
     * MyBatis-Plus的物理分页插件
     * @return
     */
     @Bean
      public PaginationInterceptor paginationInterceptor() {
          return new PaginationInterceptor();
      }

     /**
      * MyBatis-Plus的SQL执行效率分析插件
```

```
     * @return
     */
    @Bean
    public PerformanceInterceptor performanceInterceptor() {
       return new PerformanceInterceptor();
     }

}
```

9.6 代码生成器

```
package mp.demo.controller;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import com.baomidou.mybatisplus.core.toolkit.StringPool;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.InjectionConfig;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
import com.baomidou.mybatisplus.generator.config.FileOutConfig;
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.TemplateConfig;
import com.baomidou.mybatisplus.generator.config.po.TableFill;
import com.baomidou.mybatisplus.generator.config.po.TableInfo;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;
public class CodeCreater {



    public static void main(String[] args) {
        Boolean isOverRide = true;//是否覆盖原来的文件
        Boolean activeRecord = true;//是否生成支持AR模式的代码
        String parentPackageName = "mp.demo";//生成的文件的父包名
        String controller = "controller";//生成的controller的名称
        String service = "service";//生成的service的名称
        String serviceImpl = "serviceImpl";//生成的serviceImpl的名称
        String entity = "pojo";//生成的pojo的名称
        String mapper = "mapper";//生成的dao的名称
        String tableName = "student";//指定要生成的表，不指定生成全部的表
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306/boot?autoReconnect=true&useUnicode=true&characterEncoding=utf-
8";
        String user = "root";
        String password = "root";

        create(driver,url,user,password,isOverRide , activeRecord, parentPackageName, controller, service,
serviceImpl, entity, mapper, tableName);
    }
```

```
    public static void create(String driver, String url, String user, String password, Boolean isOverRide,Boolean
activeRecord,String parentPackageName,String controller,String service,String serviceImpl,String entity,String
mapper,String tableName) {
        // 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");
        // TODO 设置用户名
        gc.setAuthor("yuan");
        gc.setOpen(true);
        // service 命名方式
        gc.setServiceName("%sService");
        // service impl 命名方式
        gc.setServiceImplName("%sServiceImpl");
        // 自定义文件命名，注意 %s 会自动填充表实体属性!
        gc.setMapperName("%sMapper");
        gc.setXmlName("%sMapper");
        if(isOverRide==null) {
            gc.setFileOverride(true);
        }else {
            gc.setFileOverride(isOverRide);
        }

        gc.setActiveRecord(activeRecord);
        // XML 二级缓存
        gc.setEnableCache(false);
        // XML ResultMap
        gc.setBaseResultMap(true);
        // XML columList
        gc.setBaseColumnList(true);
        mpg.setGlobalConfig(gc);

        // TODO 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl(url);
        dsc.setDriverName(driver);
        dsc.setUsername(user);
        dsc.setPassword(password);
        mpg.setDataSource(dsc);

        // TODO 包配置
        PackageConfig pc = new PackageConfig();
        //pc.setModuleName(scanner("模块名"));
        if(parentPackageName!=null) {
            pc.setParent(parentPackageName);
        }else {
            pc.setParent("com.wwxy");
        }

        if(entity!=null) {
            pc.setEntity(entity);
        }else {
            pc.setEntity("entity");
        }

        if(service!=null) {
```

```
        pc.setService(service);
    }else {
        pc.setService("service");

    }

    if(serviceImpl!=null) {
        pc.setServiceImpl(serviceImpl);
    }else {
        pc.setServiceImpl("serviceImpl");
    }

    if(controller!=null) {
        pc.setController(controller);
    }else {
        pc.setController("controller");
    }
    mpg.setPackageInfo(pc);

    // 自定义需要填充的字段
    List<TableFill> tableFillList = new ArrayList<>();
    //如 每张表都有一个创建时间、修改时间
    //而且这基本上就是通用的了，新增时，创建时间和修改时间同时修改
    //修改时，修改时间会修改，
    //虽然像Mysql数据库有自动更新几只，但像ORACLE的数据库就没有了，
    //使用公共字段填充功能，就可以实现，自动按场景更新了。
    //如下是配置
    //TableFill createField = new TableFill("gmt_create", FieldFill.INSERT);
    //TableFill modifiedField = new TableFill("gmt_modified", FieldFill.INSERT_UPDATE);
    //tableFillList.add(createField);
    //tableFillList.add(modifiedField);

    // 自定义配置
    InjectionConfig cfg = new InjectionConfig() {
        @Override
        public void initMap() {
            // to do nothing
        }
    };
    List<FileOutConfig> focList = new ArrayList<>();
    focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {
        @Override
        public String outputFile(TableInfo tableInfo) {
            // 自定义输入文件名称
            return projectPath + "/src/main/resources/mapper/"
                    + "/" + tableInfo.getEntityName() + "Mapper" + StringPool.DOT_XML;
        }
    });
    cfg.setFileOutConfigList(focList);
    mpg.setCfg(cfg);
    mpg.setTemplate(new TemplateConfig().setXml(null));

    // 策略配置
    StrategyConfig strategy = new StrategyConfig();
    strategy.setNaming(NamingStrategy.underline_to_camel);
    strategy.setColumnNaming(NamingStrategy.underline_to_camel);
    strategy.setEntityLombokModel(true);
    // 设置逻辑删除键
    strategy.setLogicDeleteFieldName("deleted");
    // TODO 指定生成的bean的数据库表名
```

```
        if(tableName!=null) {
            strategy.setInclude(tableName);
        }
        //strategy.setSuperEntityColumns("id");
        // 驼峰转连字符
        strategy.setControllerMappingHyphenStyle(true);
        mpg.setStrategy(strategy);
        // 选择 freemarker 引擎需要指定如下加，注意 pom 依赖必须有!
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }


}
```

9.7、 MyBatisPlus设置开启swagger2支持

Mybatisplus开启swagger支持后，代码生成器生成的实体类会根据数据表中的字段注释和表的注释自动生成swagger的注解如：@ApiModel和@@ApiModelProperty,解决swagger2无法读取实体类注释来作为Api页面的接口注释的问题。

代码生成器类修改如下：

> 在全局配置类里面设置开启swagger2