MyBatis Plus2.3

# 第1章简介

## 1.1 MyBatisPlus 介绍

MyBatis-Plus(简称 简称 MP), 是一个 MyBatis 的增强 工具包，只做增强不做改变. 为简化开

发工作、提高生产率而生

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。

## 1.2 代码 及文档 发布地址

官方地址:

<http://mp.baomidou.com>

代码发布地址:

Github: <https://github.com/baomidou/mybatis-plus>

Gitee: <https://gitee.com/baomidou/mybatis-plus>

文档发布地址:

[https://baomidou.gitee.io/mybatis-plus-doc/#](https://baomidou.gitee.io/mybatis-plus-doc/" \l "/generic-crud)

## 1.3 前置知识

Mybatis

Spring

Maven

# 第2章集成 MybatisPlus

## 2.1 创建测试表

|  |
| --- |
| SET FOREIGN\_KEY\_CHECKS=0;  -- ----------------------------  -- Table structure for tbl\_employee  -- ----------------------------  DROP TABLE IF EXISTS `tbl\_employee`;  CREATE TABLE `tbl\_employee` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `last\_name` varchar(50) DEFAULT NULL,  `email` varchar(50) DEFAULT NULL,  `gender` char(1) DEFAULT NULL,  `age` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;  -- ----------------------------  -- Records of tbl\_employee  -- ----------------------------  INSERT INTO `tbl\_employee` VALUES ('1', 'Tom', 'tom@atguigu.com', '1', '22');  INSERT INTO `tbl\_employee` VALUES ('2', 'Jerry', 'jerry@atguigu.com', '0', '25');  INSERT INTO `tbl\_employee` VALUES ('3', 'Black', 'black@atguigu.com', '1', '30');  INSERT INTO `tbl\_employee` VALUES ('4', 'White', 'white@atguigu.com', '0', '35'); |

## 2.2 创建 javaBean

|  |
| --- |
| **package** xia.wenjie.po;  **import** com.baomidou.mybatisplus.annotations.TableName;  */\*\*  \** ***@描述:*** *\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/14  \*/* @TableName(value = **"tbl\_employee"**) **public class** Employee {  **private** Integer **id**; *// int(11) NOT NULL AUTO\_INCREMENT,* **private** String **last\_name**; *// varchar(50) DEFAULT NULL,* **private** String **email**; *// varchar(50) DEFAULT NULL,* **private** Integer **gender**; *// char(1) DEFAULT NULL,* **private** Integer **age**; *// int(11) DEFAULT NULL,* **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** String getLast\_name() {  **return last\_name**;  }   **public void** setLast\_name(String last\_name) {  **this**.**last\_name** = last\_name;  }   **public** String getEmail() {  **return email**;  }   **public void** setEmail(String email) {  **this**.**email** = email;  }   **public** Integer getGender() {  **return gender**;  }   **public void** setGender(Integer gender) {  **this**.**gender** = gender;  }   **public** Integer getAge() {  **return age**;  }   **public void** setAge(Integer age) {  **this**.**age** = age;  }   @Override  **public** String toString() {  **return "Employee{"** +  **"id="** + **id** +  **", last\_name='"** + **last\_name** + **'\''** +  **", email='"** + **email** + **'\''** +  **", gender="** + **gender** +  **", age="** + **age** +  **'}'**;  } } |

## 2.3 依赖配置

### 1) 在 pom.xml 中加入对 MP、Spring、连接池、Junit、Mysql 驱动等依赖

|  |
| --- |
| <**dependencies**>  *<!-- 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**>8.0.13</**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**> </**dependencies**> |

特别说明: 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！

### 加入 MyBatis 的全局配置文件

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**>  </**configuration**> |

### 加入 log4j.xml

|  |
| --- |
| *<?***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**> |

### 加入 db.properties

|  |
| --- |
| **jdbc.driver**=**com.mysql.jdbc.Driver jdbc.url**=**jdbc:mysql://localhost:3306/mp jdbc.username**=**root jdbc.password**=**root** |

### 加入 spring 的配置文件 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">-->* <**bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!-- 别名处理 -->* <**property name="typeAliasesPackage" value="xia.wenjie.po"**></**property**>  </**bean**>  *<!--  配置 mybatis 扫描 mapper 接口的路径  -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**>  </**bean**> </**beans**> |

## 2.4测试

### 测试spring-mybatis 的环境 保证ok

|  |
| --- |
| **package** xia.wenjie.test;  **import** java.sql.Connection; **import** java.util.List;   **import** javax.sql.DataSource;  **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** xia.wenjie.mapper.EmployeeMapper; **import** xia.wenjie.po.Employee;   */\*\*  \** ***@描述:*** *\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/14  \*/* **public class** DemoMybatispius {  **private** ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);  @Test  **public void** testEnvironment() **throws** Exception {  DataSource ds = **iocContext**.getBean(**"dataSource"**, DataSource.**class**);  Connection conn = ds.getConnection();  System.***out***.println(conn+**"-------------"**);  } } |

## 2.5 集成 MP

Mybatis-Plus 的集成非常简单，对于 Spring，我们仅仅需要把 Mybatis 自带的

MybatisSqlSessionFactoryBean 替换为 MP 自带的即可。

|  |
| --- |
| *<!-- 配置 SqlSessionFactoryBean --> <!--<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">-->* <**bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!-- 别名处理 -->* <**property name="typeAliasesPackage" value="xia.wenjie.po"**></**property**> </**bean**> |

# 第3章入门 HelloWorld

## 3.1 通用 CRUD

1) 提出问题:

假设我们已存在一张 tbl\_employee 表，且已有对应的实体类 Employee，实现

tbl\_employee 表的 CRUD 操作我们需要做什么呢？

2) 实现方式:

基于 Mybatis

需要编写 EmployeeMapper 接口，并手动编写 CRUD 方法

提供 EmployeeMapper.xml 映射文件，并手动编写每个方法对应的 SQL 语句.

基于 MP

只需要创建 EmployeeMapper 接口, 并继承 BaseMapper

|  |
| --- |
| **package** xia.wenjie.mapper;  **import** com.baomidou.mybatisplus.mapper.BaseMapper; **import** xia.wenjie.po.Employee; */\*\*  \** ***@描述:*** *BaseMapper的泛型就是操作的实体类型**\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/14  \*/* **public interface** EmployeeMapper **extends** BaseMapper<Employee>{ } |

如果需要mybatis-plus源码中有中文注释，需要加下面的依赖再下载源码。

|  |
| --- |
| *<!-- https://mvnrepository.com/artifact/com.baomidou/mybatis-plus-boot-starter -->* <**dependency**>  <**groupId**>com.baomidou</**groupId**>  <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  <**version**>2.3</**version**> </**dependency**> |

## 3.2 插入操作insert方法

|  |
| --- |
| **package** xia.wenjie.test;  **import** java.sql.Connection; **import** java.util.List; **import** javax.sql.DataSource; **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** xia.wenjie.mapper.EmployeeMapper; **import** xia.wenjie.po.Employee;**public class** DemoMybatispius {  **private** ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);   **private** EmployeeMapper **employeeMapper** = **iocContext**.getBean(**"employeeMapper"**, EmployeeMapper.**class**); @Test  **public void** insertTest() {  Employee employee = **new** Employee();  employee.setLast\_name(**"xiawenjie"**);  employee.setEmail(**"131@qq.com"**);  employee.setGender(1);  employee.setAge(30);  **employeeMapper**.insert(employee);  }} |

报错：数据库的主键设置的是自增的

|  |
| --- |
| Caused by: org.apache.ibatis.reflection.ReflectionException: Could not set property 'id' of 'class xia.wenjie.po.Employee' with value '1184012418776522753' Cause: java.lang.IllegalArgumentException: argument type mismatch |

[https://baomidou.gitee.io/mybatis-plus-doc/#/generic-crud?id=%e7%ae%80%e5%8d%95%e4%bb%8b%e7%bb%8d](https://baomidou.gitee.io/mybatis-plus-doc/" \l "/generic-crud?id=%e7%ae%80%e5%8d%95%e4%bb%8b%e7%bb%8d)

|  |
| --- |
| **public class** Employee {  */\*  \* @TableId:  \* value: 指定表中的主键列的列名， 如果实体属性名与列名一致，可以省略不指定.  \* type: 指定主键策略.  \*/* @TableId(value = **"id"**, type = IdType.***AUTO***)  **private** Integer **id**; *// int(11) NOT NULL AUTO\_INCREMENT,* **private** String **last\_name**; *// varchar(50) DEFAULT NULL,* **private** String **email**; *// varchar(50) DEFAULT NULL,* **private** Integer **gender**; *// char(1) DEFAULT NULL,* **private** Integer **age**; *// int(11) DEFAULT NULL,* |

再运行报 Table 'mp.employee' doesn't exist

默认在mp库找名字为employee 的表

|  |
| --- |
| @TableName(value = **"tbl\_employee"**) **public class** Employee { |

运行 插入成功。

查看打印的sql

|  |
| --- |
| Preparing: INSERT INTO tbl\_employee ( last\_name, email, gender, age ) VALUES ( ?, ?, ?, ? ) |

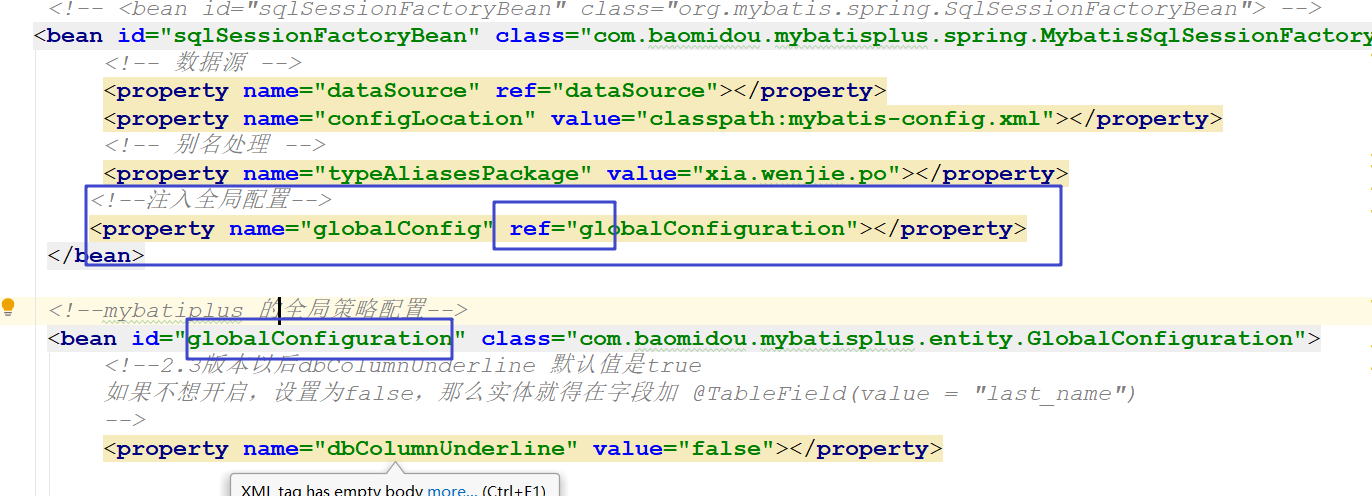
### 全局策略配置

现在把实体的last\_name改为驼峰命名的 lastName，插入数据还是成功。 （mybatis-config.xml 配置文件里没有多字段驼峰的处理）

是因为mybatiplus2.3后版本的全局配置已经默认配置已经开启了下划线命名转换驼峰了.可以不配置

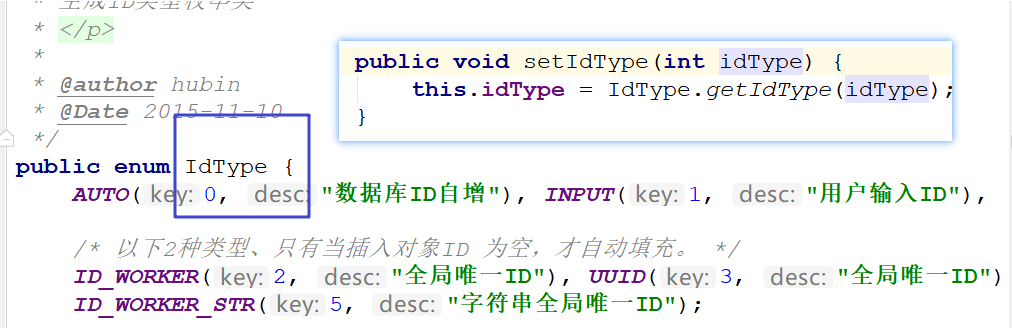
|  |
| --- |
| *<!--mybatiplus 的全局策略配置-->* <**bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration"**>  *<!--2.3版本以后dbColumnUnderline 默认值是true  如果不想开启，设置为false，那么实体就得在字段加 @TableField(value = "last\_name")  -->* <**property name="dbColumnUnderline" value="true"**></**property**> </**bean**> |

全局策略配置生效需要注入到**MybatisSqlSessionFactoryBean**



### 去掉实体上的@TableName， @TableId

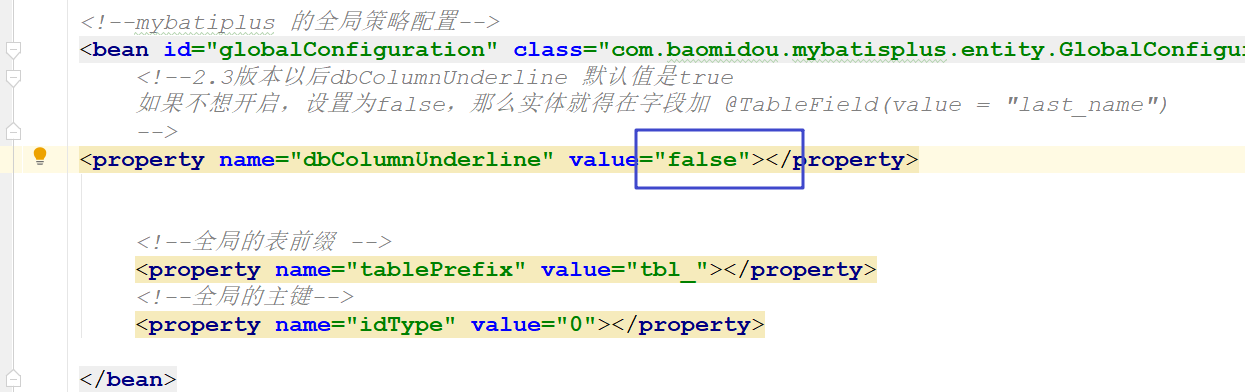
如果有很多的表需要管理，进行配置就很麻烦了，可以在全局配置



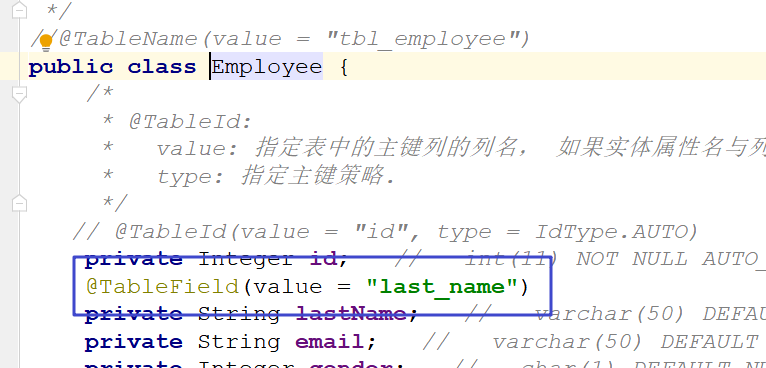
|  |
| --- |
| *<!--mybatiplus 的全局策略配置-->* <**bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration"**>  *<!--2.3版本以后dbColumnUnderline 默认值是true  如果不想开启，设置为false，那么实体就得在字段加 @TableField(value = "last\_name")  -->  <!-- <property name="dbColumnUnderline" value="true"></property>-->    <!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**>  *<!--全局的主键-->* <**property name="idType" value="0"**></**property**>  </**bean**> |

### @TableField讲解

关闭*使用下划线命名*

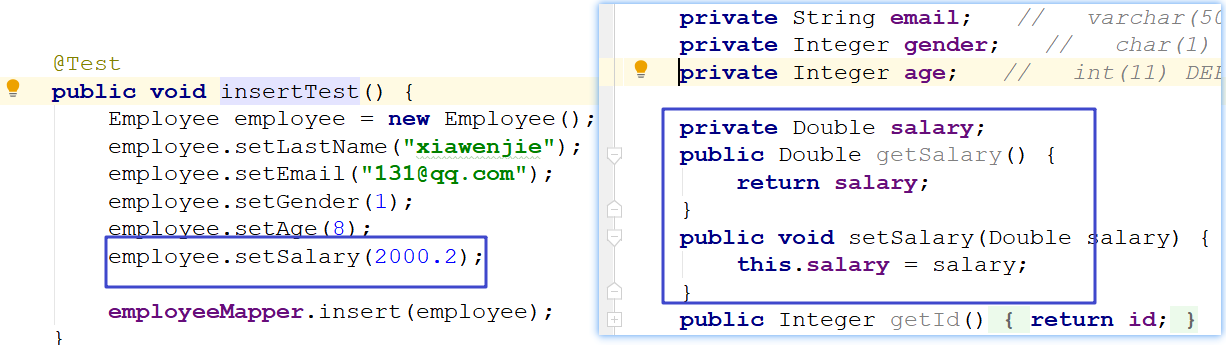


实体就要写表的字段了



如果实体添加一个字段，但数据库不存进去，把它设置成不是数据库字段

插入数据会报：Unknown column 'salary' in 'field list'



|  |
| --- |
| **public class** Employee {  */\*  \* @TableId:  \* value: 指定表中的主键列的列名， 如果实体属性名与列名一致，可以省略不指定.  \* type: 指定主键策略.  \*/  // @TableId(value = "id", type = IdType.AUTO)* **private** Integer **id**; *// int(11) NOT NULL AUTO\_INCREMENT,* @TableField(value = **"last\_name"**)  **private** String **lastName**; *// varchar(50) DEFAULT NULL,* **private** String **email**; *// varchar(50) DEFAULT NULL,* **private** Integer **gender**; *// char(1) DEFAULT NULL,* **private** Integer **age**; *// int(11) DEFAULT NULL,   /\*  \* 否为数据库表字段  \* 默认 true 存在，false 不存在  \*/* @TableField(exist = **false**)  **private** Double **salary**;   **public** Double getSalary() {  **return salary**;  }   **public void** setSalary(Double salary) {  **this**.**salary** = salary;  } |

### 插入数据获取主键值 支持主键自增的数据库

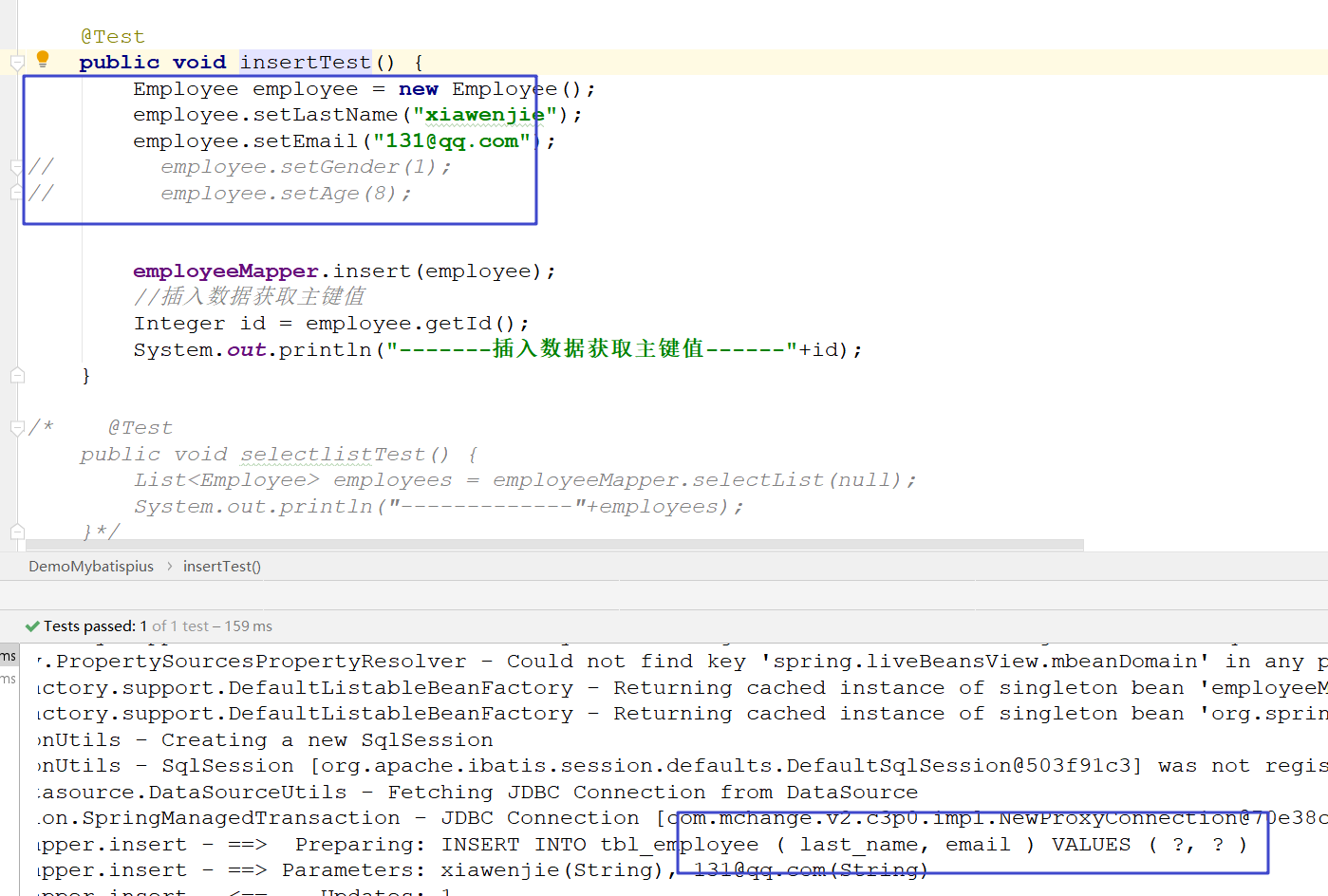
MybatisPlus: 自动将主键值回写到实体类中

|  |
| --- |
| @Test **public void** insertTest() {  Employee employee = **new** Employee();  employee.setLastName(**"xiawenjie"**);  employee.setEmail(**"131@qq.com"**);  employee.setGender(1);  employee.setAge(8);  employee.setSalary(2000.2);   **employeeMapper**.insert(employee);  *//插入数据获取主键值* Integer id = employee.getId();  System.***out***.println(**"-------插入数据获取主键值------"**+id); } |

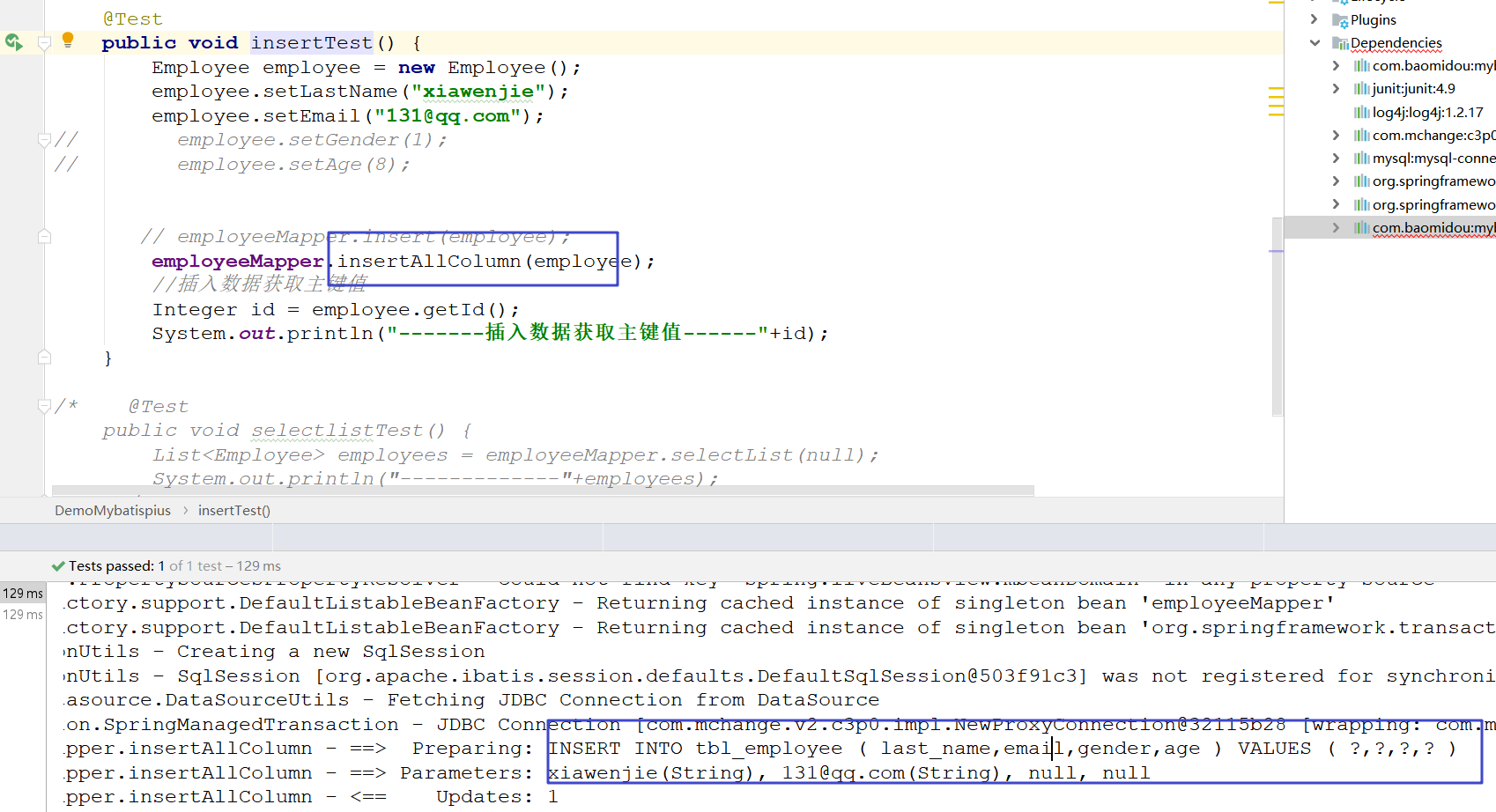
Mybatis: 需要通过 useGeneratedKeys 以及 keyProperty 来设置

### Integer insertAllColumn(T entity)

实体少几个属性赋值，或设置属性值为null,对应的sql插入字段也会少，如果想插入所有的字段，就用insertAllColumn







总结：

*插入到数据库  
insert方法在插入时， 会根据实体类的每个属性进行非空判断，只有非空的属性对应的字段才会出现到SQL语句中  
insertAllColumn方法在插入时， 不管属性是否非空， 属性所对应的字段都会出现到SQL语句中.*

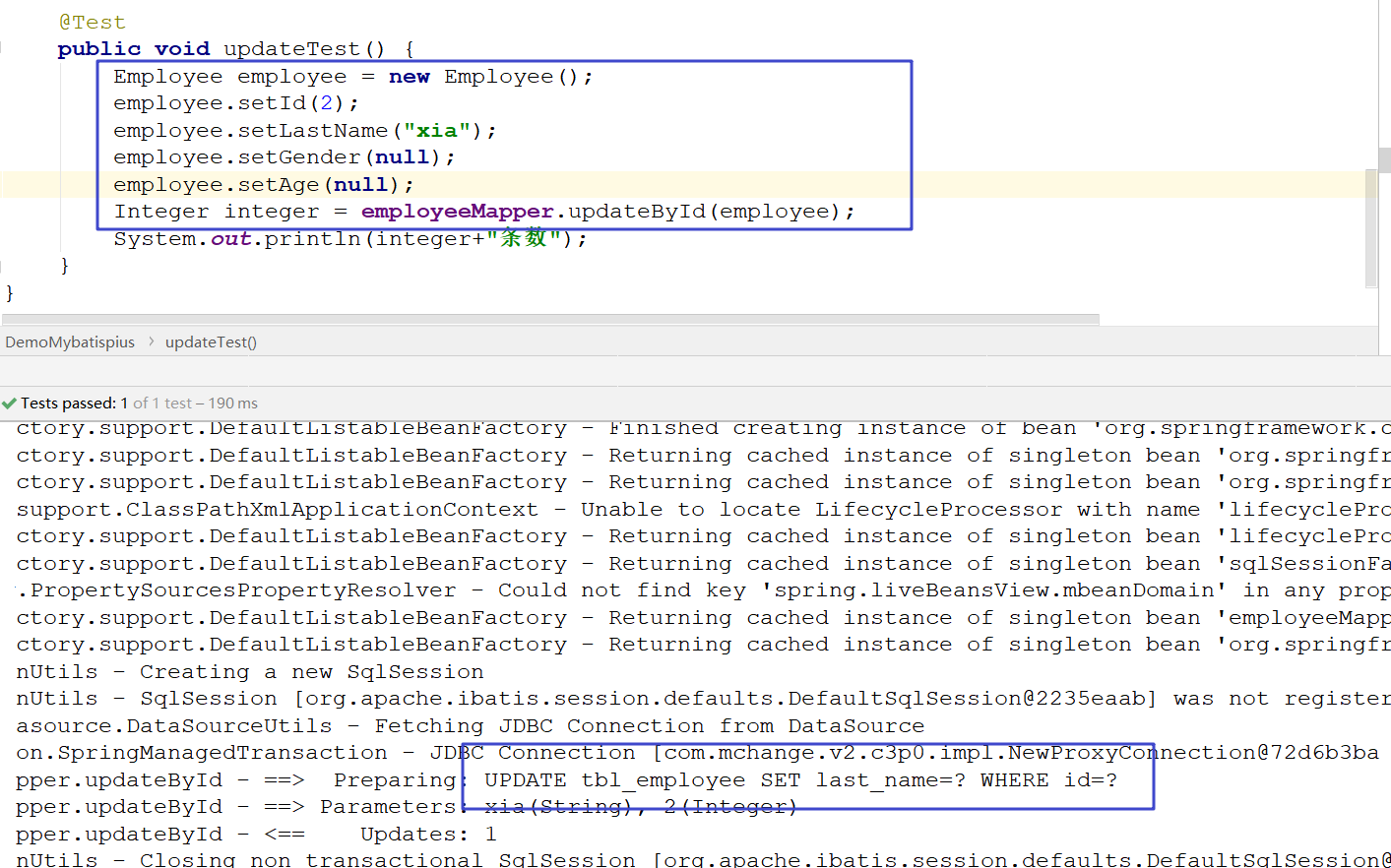
## 3.3 更新操作

1. Integer updateById(@Param("et") T entity);

*会根据实体类的每个属性进行非空判断，只有非空的属性对应的字段才会出现到SQL语句中*

2)Integer updateAllColumnById(@Param("et") T entity)

*不管属性是否非空， 属性所对应的字段都会出现到SQL语句中*



## 3.4 查询操作

#### 1)T selectById(Serializable id); 根据 ID 查询 主键ID

#### 2)T selectOne(@Param("ew") T entity); 根据 entity 条件，查询一条记录

有查到多条记录会报：Expected one result (or null) to be returned by selectOne(), but found: 8

|  |
| --- |
| @Test **public void** selectTest() {  Employee employee = **new** Employee();  employee.setLastName(**"xiawenjie"**);  employee.setGender(3);  *//SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE last\_name=? AND gender=?* Employee emplo = **employeeMapper**.selectOne(employee);  System.***out***.println(emplo); } |

#### 3)List<T> selectBatchIds(List<? extends Serializable> idList); 查询（根据ID 批量查询）主键ID列表

|  |
| --- |
| @Test **public void** selectTest() {  ArrayList<Integer> list = **new** ArrayList<Integer>();  list.add(1);  list.add(8);  *//SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE id IN ( ? , ? )*  List<Employee> employees = **employeeMapper**.selectBatchIds(list);  **for** (Employee employee : employees) {  System.***out***.println(employee);  } } |

#### 4)List<T> selectByMap(@Param("cm") Map<String, Object> columnMap); 查询（根据 columnMap 条件）

|  |
| --- |
| */\*\*  \* 查询  \*/* @Test **public void** selectTest() {  Map<String, Object> hashMap = **new** HashMap<String, Object>();  *//数据库的字段名字* hashMap.put(**"last\_name"**, **"xia"**);  hashMap.put(**"id"**, 15);  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE last\_name = ? AND id = ?* List<Employee> employees = **employeeMapper**.selectByMap(hashMap);  **for** (Employee employee : employees) {  System.***out***.println(employee);  } } |

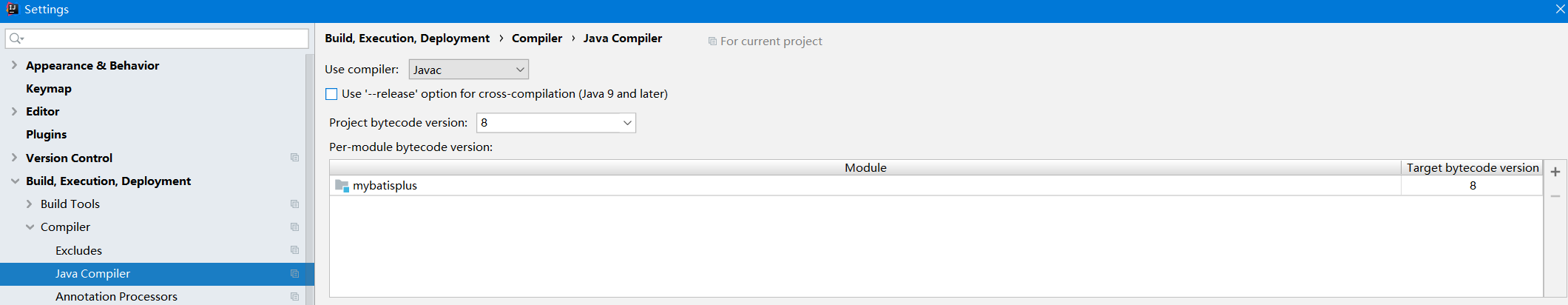
#### 5)List<T> selectPage(RowBounds rowBounds, @Param("ew") Wrapper<T> wrapper); 根据 entity 条件，查询全部记录（并翻页）

|  |
| --- |
| @Test **public void** selectTest() {  *//这里用Page,后面的可以不传  //使用的是内存分页  // SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee* List<Employee> employees = **employeeMapper**.selectPage(**new** Page<>(2,2),**null**);  **for** (Employee employee : employees) {  System.***out***.println(employee);  } } |

idea提示：Diamond types are not supported at language level '5'

报：Information:javac 1.8.0\_211 was used to compile java sources

file--> settings--> Java Compiler --- > 找到项目module，修改为1.8 就可以了。



## 3.5 删除操作

#### 1)Integer deleteById(Serializable id); 根据 ID 删除

|  |
| --- |
| */\*\*  \* 删除  \*/* @Test **public void** delectTest() {  *// DELETE FROM tbl\_employee WHERE id=?* Integer integer = **employeeMapper**.deleteById(19);  System.***out***.println(integer); } |

#### Integer deleteByMap(@Param("cm") Map<String, Object> columnMap); columnMap 条件，删除记录

|  |
| --- |
| @Test **public void** delectTest() {  Map<String, Object> hashMap = **new** HashMap<>();  *//表里面字段的名字 作为key* hashMap.put(**"last\_name"**, **"xiawenjie"**);  hashMap.put(**"gender"**, 4);  *// DELETE FROM tbl\_employee WHERE gender = ? AND last\_name = ?* Integer integer = **employeeMapper**.deleteByMap(hashMap);  System.***out***.println(integer); } |

#### Integer deleteBatchIds(List<? extends Serializable> idList); 删除（根据ID 批量删除）

|  |
| --- |
| */\*\*  \* 删除  \*/* @Test **public void** delectTest() {  List<Integer> list = **new** ArrayList<>();  list.add(17);  list.add(18);  list.add(19);  *// DELETE FROM tbl\_employee WHERE id IN ( ? , ? , ? )* Integer integer = **employeeMapper**.deleteBatchIds(list);  System.***out***.println(integer); } |

## 3.6 MybatisPlus 启动注入 SQL 原理分析

#### 1) 问题:

xxxMapper 继承了 BaseMapper<T>, BaseMapper 中提供了通用的 CRUD 方法,

方法来源于 BaseMapper, 有方法就必须有 SQL, 因为 MyBatis 最终还是需要通过

SQL 语句操作数据.

前置知识:MyBatis 源码中比较重要的一些对象， MyBatis 框架的执行流程

Configuration

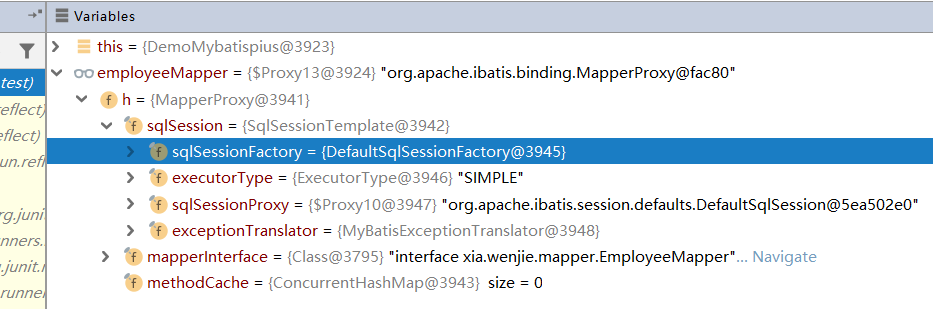
MappedStatement

……..

#### 2) 通过现象看到本质

A． employeeMapper 的本质 org.apache.ibatis.binding.MapperProxy

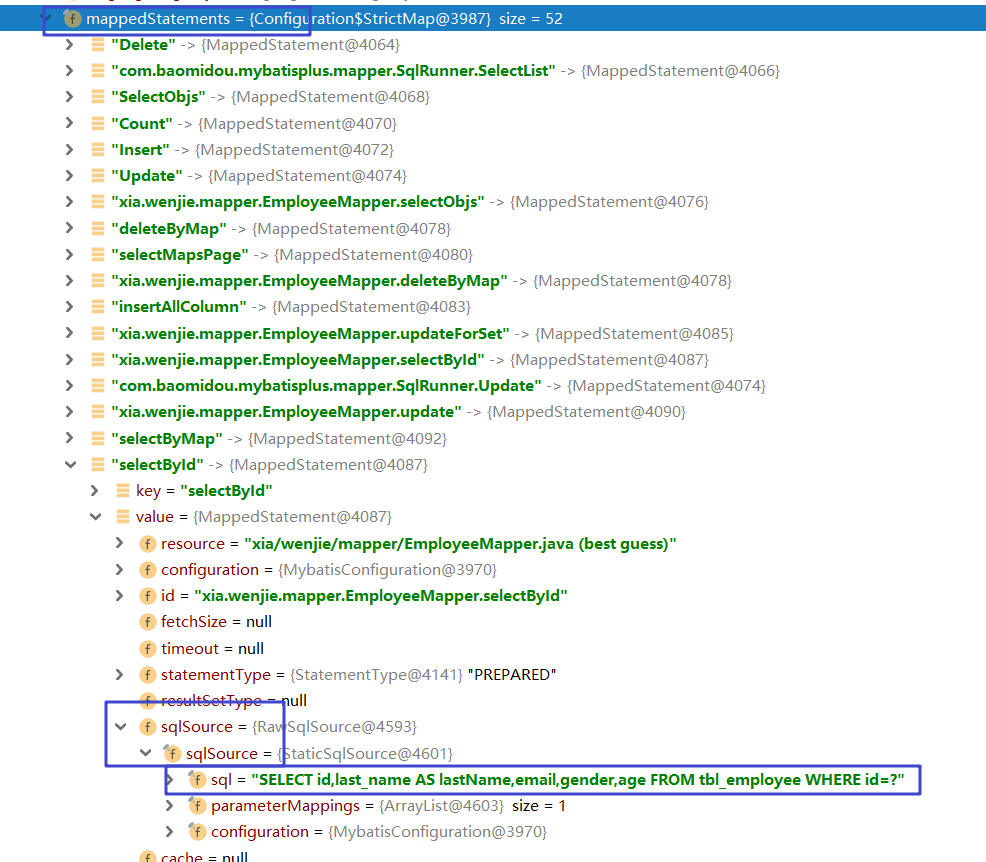
B． MapperProxy 中 sqlSession –>SqlSessionFactory



C． SqlSessionFacotry 中 → Configuration→ MappedStatements

每一个 mappedStatement 都表示 Mapper 接口中的一个方法与 Mapper 映射文件

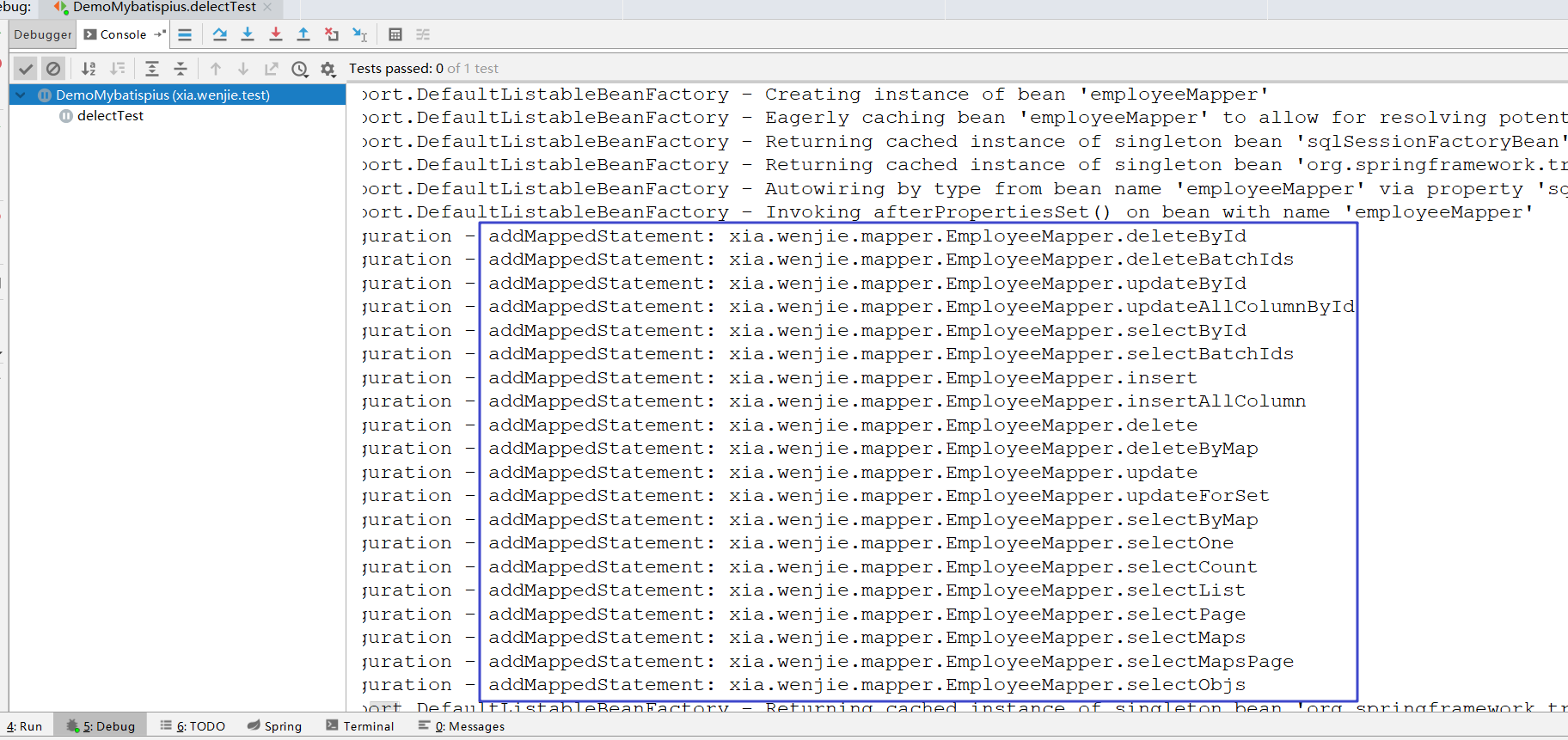
中的一个 SQL。



MP 在启动就会挨个分析 xxxMapper 中的方法，并且将对应的 SQL 语句处理好，保

存到 configuration 对象中的 mappedStatements 中.

D． 本质:



Configuration： MyBatis 或者 MP 全局配置对象

MappedStatement：一个 MappedStatement 对象对应 Mapper 配置文件中的一个

select/update/insert/delete 节点，主要描述的是一条 SQL 语句

SqlMethod : 枚举对象 ，MP 支持的 SQL 方法

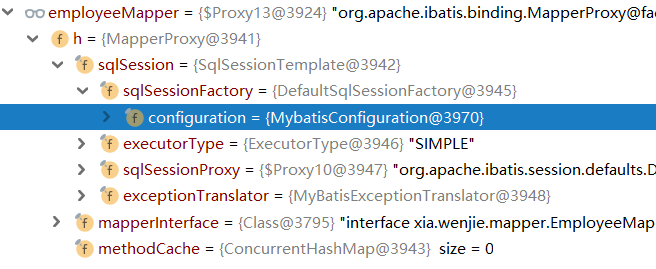
TableInfo：数据库表反射信息 ，可以获取到数据库表相关的信息

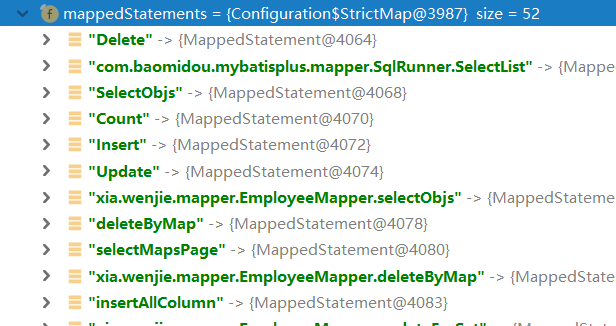
SqlSource: SQL 语句处理对象

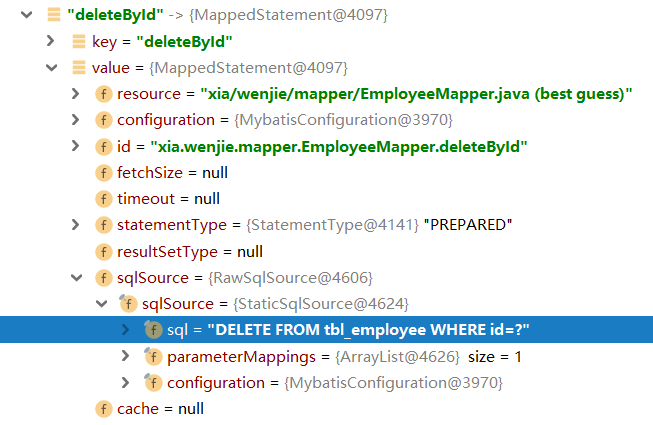
MapperBuilderAssistant： 用于缓存、SQL 参数、查询方剂结果集处理等.

通过 MapperBuilderAssistant 将每一个 mappedStatement

添加到 configuration 中的 mappedstatements 中







## 3.7通用 CRUD 小结

1) 以上是基本的 CRUD 操作，如您所见，我们仅仅需要继承一个 BaseMapper 即可实现

大部分单表 CRUD 操作。BaseMapper 提供了多达 17 个方法给大家使用, 可以极其方

便的实现单一、批量、分页等操作。极大的减少开发负担，难道这就是 MP 的强大之处

了吗？

2) 提出需求:

现有一个需求，我们需要分页查询 tbl\_employee 表中，年龄在 18~50 之间性别为男且

姓名为 xx 的所有用户，这时候我们该如何实现上述需求呢？

MyBatis : 需要在 SQL 映射文件中编写带条件查询的 SQL,并基于 PageHelper 插件完成

分页. 实现以上一个简单的需求，往往需要我们做很多重复单调的工作。普通的 Mapper

能够解决这类痛点吗？

MP: 依旧不用编写 SQL 语句, MP 提供了功能强大的条件构造器 EntityWrapper

# 第4章条件构造器 EntityWrapper

## 4.1 EntityWrapper 简介

1) Mybatis-Plus 通过 EntityWrapper（简称 EW，MP 封装的一个查询条件构造器）或者

Condition（与 EW 类似） 来让用户自由的构建查询条件，简单便捷，没有额外的负担，

能够有效提高开发效率

2)实体包装器，主要用于处理 sql 拼接，排序，实体参数查询等

3)注意: 使用的是数据库字段，不是 Java 属性!

4)条件参数说明:

| **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") |

## 4.2 使用 EntityWrapper 的方式打开如上需求:

|  |
| --- |
| */\*\*  \* 分页查询 tbl\_employee 表中，  \* 年龄在 18~50 之间  \* 性别为男且  \* 姓名为 xx 的所有用户  \*/* @Test **public void** selectTest() {  *//内存分页  // SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name = ? AND gender = ? AND age BETWEEN ? AND ?)* List<Employee> employees = **employeeMapper**.selectPage(**new** Page<>(0, 20), **new** EntityWrapper<Employee>().eq(**"last\_name"**, **"xiawenjie"**).eq(**"gender"**, 1).between(**"age"**, 8, 30));  **for** (Employee employee : employees) {  System.***out***.println(employee);  } } |

## 4.3 带条件的查询

1) List<T> selectList(@Param("ew") Wrapper<T> wrapper);

|  |
| --- |
| @Test **public void** selectTest() {  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name LIKE ?)  // ==> Parameters: %xia%(String)* List<Employee> employees = **employeeMapper**.selectList(**new** EntityWrapper<Employee>().like(**"last\_name"**, **"xia"**));  System.***out***.println(employees); } |

## 4.4 带条件的修改

1) Integer update(@Param("et") T entity, @Param("ew") Wrapper<T> wrapper);

|  |
| --- |
| @Test  **public void** UpdateTest() {  Employee employee = **new** Employee();  employee.setLastName(**"鬼谷子"**);  employee.setAge(120); *// UPDATE tbl\_employee SET last\_name=?, age=? WHERE (last\_name = ?) // ==> Parameters: 鬼谷子(String), 120(Integer), 马云(String)* Integer integer = **employeeMapper**.update(employee, **new** EntityWrapper<Employee>().eq(**"last\_name"**, **"马云"**));  } |

## 4.5 带条件的删除

1) Integer delete(@Param("ew") Wrapper<T> wrapper);

|  |
| --- |
| @Test **public void** DeleteTest() {  *// DELETE FROM tbl\_employee WHERE (last\_name = ? AND gender = ?)  // ==> Parameters: xiawenjie(String), 2(Integer)* Integer integer = **employeeMapper**.delete(**new** EntityWrapper<Employee>().eq(**"last\_name"**, **"xiawenjie"**).eq(**"gender"**, 2));  System.***out***.println(integer); } |

## 4.6使用Condition 的方式打开如上需求

|  |
| --- |
| */\*\*  \* 分页查询 tbl\_employee 表中，  \* 年龄在 18~50 之间  \* 性别为男且  \* 姓名为 xx 的所有用户  \*/* @Test  **public void** ConditionTest() {  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name = ? AND gender = ? AND age BETWEEN ? AND ?) // ==> Parameters: xiawenjie(String), 1(Integer), 8(Integer), 30(Integer) // SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name = ? AND gender = ? AND age BETWEEN ? AND ?)   // List<Employee> employees = employeeMapper.selectPage(new Page<>(1,20),Condition.create().eq("last\_name","xiawenjie").eq("gender",1).between("age",8,30));* List<Employee> employees = **employeeMapper**.selectPage(**new** Page<>(1,20),**new** Condition().eq(**"last\_name"**,**"xiawenjie"**).eq(**"gender"**,1).between(**"age"**,8,30));  System.***out***.println(employees);  } |

## 4.7 小结

MP: EntityWrapper Condition 条件构造器

MyBatis MBG : xxxExample→Criteria : QBC( Query By Criteria)

Hibernate 、 通用 Mapper

# 第5章：ActiveRecord(活动记录)

Active Record(活动记录)，是一种领域模型模式，特点是一个模型类对应关系型数据库中的

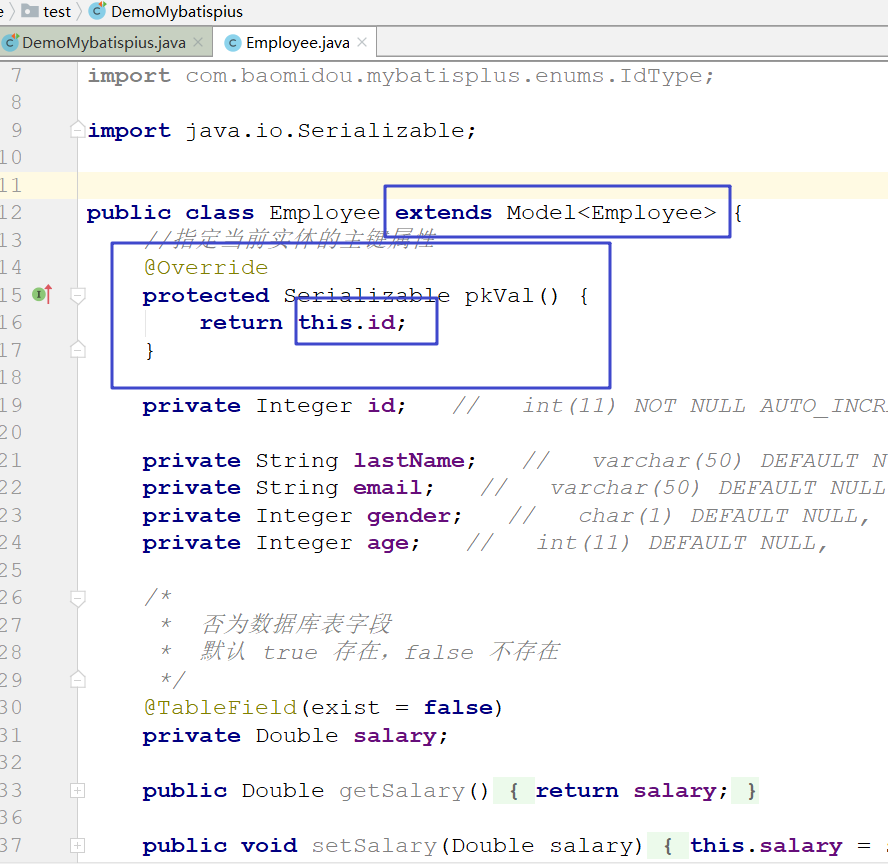
一个表，而模型类的一个实例对应表中的一行记录。

ActiveRecord 一直广受动态语言（ PHP 、 Ruby 等）的喜爱，而 Java 作为准静态语言，

对于 ActiveRecord 往往只能感叹其优雅，所以 MP 也在 AR 道路上进行了一定的探索

## 5.1 如何使用Active Record

仅仅需要让实体类继承 Model 类且实现主键指定方法，即可开启 AR 之旅



## 5.2 Active Record基本CRUD

### 1)插入操作

public boolean insert() insertAllColumn()*插入（所有字段插入）*

一样的套路

|  |
| --- |
| **package** xia.wenjie.test; **import** java.sql.Connection; **import** java.util.ArrayList; **import** java.util.HashMap; **import** java.util.List; **import** java.util.Map; **import** javax.sql.DataSource; **import** com.baomidou.mybatisplus.mapper.Condition; **import** com.baomidou.mybatisplus.mapper.EntityWrapper; **import** com.baomidou.mybatisplus.plugins.Page; **import** org.apache.ibatis.session.RowBounds; **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** xia.wenjie.mapper.EmployeeMapper; **import** xia.wenjie.po.Employee; */\*\***\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/14  \*/* **public class** DemoMybatispius {  **private** ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);  *//private EmployeeMapper employeeMapper = iocContext.getBean("employeeMapper", EmployeeMapper.class);*  @Test  **public void** insertTest() {  *// 初始化 成功标识* **boolean** result = **false**;   Employee employee = **new** Employee();  employee.setLastName(**"xiawenjie"**);  employee.setEmail(**"13111@qq.com"**);  employee.setGender(**null**);  employee.setAge(**null**);  *// INSERT INTO tbl\_employee ( last\_name, email ) VALUES ( ?, ? )  // ==> Parameters: xiawenjie(String), 13111@qq.com(String)* **boolean** insert = employee.insert();  System.***out***.println(insert); *//true* } } |

### 2)修改操作

public boolean updateById()

|  |
| --- |
| @Test **public void** updateTest() {  *// 初始化 成功标识* **boolean** result = **false**;   Employee employee = **new** Employee();  employee.setLastName(**"xiawenjie"**);  employee.setEmail(**"1311111111@qq.com"**);  employee.setGender(**null**);  employee.setAge(**null**);  employee.setId(22); *// UPDATE tbl\_employee SET last\_name=?, email=? WHERE id=?* **boolean** insert = employee.updateById();  System.***out***.println(insert); *//true* } |

### 3)查询操作

public T selectById()

|  |
| --- |
| @Test **public void** selectById() {  Employee employee = **new** Employee();  employee.setId(12);  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE id=?* Employee employee1 = employee.selectById();  System.***out***.println(employee1); } |

public T selectById(Serializable id)

|  |
| --- |
| @Test **public void** selectById() {  Employee employee = **new** Employee();  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE id=?* Employee employee1 = employee.selectById(12);  System.***out***.println(employee1); } |

public List<T> selectAll()

|  |
| --- |
| @Test **public void** selectAllTest() {  Employee employee = **new** Employee();  employee.setLastName(**"xiawenjie"**); *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee* List<Employee> employees = employee.selectAll();  System.***out***.println(employees);  } |

public List<T> selectList(Wrapper wrapper)

|  |
| --- |
| @Test  **public void** selectList() {  Employee employee = **new** Employee(); *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name = ?)* List<Employee> employees = employee.selectList(**new** EntityWrapper().eq(**"last\_name"**, **"xiawenjie"**));  System.***out***.println(employees);  } |

public int selectCount(Wrapper wrapper)

|  |
| --- |
| @Test  **public void** selectCount() {  Employee employee = **new** Employee(); *// SELECT COUNT(1) FROM tbl\_employee WHERE (last\_name = ?)* **int** i = employee.selectCount(**new** EntityWrapper().eq(**"last\_name"**, **"xiawenjie"**));  System.***out***.println(i);  } |

### 4)删除操作

一样的套路

public boolean deleteById()

Public boolean deleteById(Serializable id)

public boolean delete(Wrapper wrapper)

### 5)分页复杂操作

public Page<T> selectPage(Page<T> page, Wrapper<T> wrapper)

|  |
| --- |
| @Test **public void** selectPage() {  Employee employee = **new** Employee();  *// SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee WHERE (last\_name = ?)   //得到的是page对象* Page<Employee> page = employee.selectPage(**new** Page<>(2, 4),  **new** EntityWrapper()  .eq(**"last\_name"**, **"xiawenjie"**));    *// Page:{ [Pagination { total=0 ,size=4 ,pages=0 ,current=2 }], records-size:4 }* System.***out***.println(page);  List<Employee> employees = page.getRecords();  System.***out***.println(employees); } |

## 5.3 ActiveRecord 小结

ActiveRecord 模式提供了一种更加便捷的方式实现 CRUD 操作，其本质还是调用的 Mybatis 对应的方法，类似于语法糖

语法糖是指计算机语言中添加的某种语法，这种语法对原本语言的功能并没有影响.可以更方便开发者使用，可以避免出错的机会，让程序可读性更好.

到此，我们简单领略了 Mybatis-Plus 的魅力与高效率，值得注意的一点是：我们提供了强大的代码生成器，可以快速生成各类代码，真正的做到了即开即用

# 第6章：代码生成器

MP 提供了大量的自定义设置，生成的代码完全能够满足各类型的需求

MP 的代码生成器 和 Mybatis MBG 代码生成器:

MP 的代码生成器都是基于 java 代码来生成。MBG 基于 xml 文件进行代码生成 MyBatis 的代码生成器可生成: 实体类、Mapper 接口、Mapper 映射文件

MP 的代码生成器可生成: 实体类(可以选择是否支持 AR)、Mapper 接口、Mapper 映射文件、 Service 层、Controller 层.

表及字段命名策略选择

在MP 中，我们建议数据库表名 和 表字段名采用驼峰命名方式， 如果采用下划线命名方式 请开启全局下划线开关，如果表名字段名命名方式不一致请注解指定，我们建议最好保持一致。

这么做的原因是为了避免在对应实体类时产生的性能损耗，这样字段不用做映射就能直接和实体类对应。当然如果项目里不用考虑这点性能损耗，那么你采用下滑线也是没问题的，只需要在生成代码时配置 dbColumnUnderline 属性就可以

## 6.1 代码生成器依赖

### 1)模板引擎

MP 的代码生成器默认使用的是 Apache 的 Velocity 模板，当然也可以更换为别的模板技术，例如 freemarker。此处不做过多的介绍。

需要加入 Apache Velocity 的依赖

|  |
| --- |
| <**dependency**>  <**groupId**>org.apache.velocity</**groupId**>  <**artifactId**>velocity-engine-core</**artifactId**>  <**version**>2.0</**version**> </**dependency**> |

### 加入slf4j ,查看日志输出信息

|  |
| --- |
| <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-api</**artifactId**>  <**version**>1.7.7</**version**> </**dependency**> <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  <**version**>1.7.7</**version**> </**dependency**> |

## 6.2 MP 代码生成器示例代码

|  |
| --- |
| **package** xia.wenjie.test;  **import** com.baomidou.mybatisplus.enums.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.DbType; **import** com.baomidou.mybatisplus.generator.config.rules.NamingStrategy; **import** org.junit.Test;  **public class** DemoMybatispius {   @Test  **public void** testGenerator() {  *//1. 全局配置* GlobalConfig config = **new** GlobalConfig();  config.setActiveRecord(**true**) *// 是否支持AR模式* .setAuthor(**"xiawenjie"**) *// 作者* .setOutputDir(**"D:\\test"**) *// 生成路径* .setFileOverride(**true**) *// 文件覆盖* .setIdType(IdType.***AUTO***) *// 主键策略* .setServiceName(**"%sService"**) *// 设置生成的service接口的名字的首字母是否为I  // IEmployeeService* .setBaseResultMap(**true**)  .setBaseColumnList(**true**);   *//2. 数据源配置* DataSourceConfig dsConfig = **new** DataSourceConfig();  dsConfig.setDbType(DbType.***MYSQL***) *// 设置数据库类型* .setDriverName(**"com.mysql.jdbc.Driver"**)  .setUrl(**"jdbc:mysql://localhost:3306/mp?serverTimezone=GMT%2B8"**)  *//时区报错需要加上 ?serverTimezone=GMT%2B8"*  .setUsername(**"root"**)  .setPassword(**"root"**);   *//3. 策略配置* StrategyConfig stConfig = **new** StrategyConfig();  stConfig.setCapitalMode(**true**) *//全局大写命名* .setDbColumnUnderline(**true**) *// 指定表名 字段名是否使用下划线* .setNaming(NamingStrategy.***underline\_to\_camel***) *// 数据库表映射到实体的命名策略* .setTablePrefix(**"tbl\_"**)  .setInclude(**"tbl\_employee"**); *// 生成的表   //4. 包名策略配置* PackageConfig pkConfig = **new** PackageConfig();  pkConfig.setParent(**"xia.wenjie"**)  .setMapper(**"mapper"**)  .setService(**"service"**)  .setController(**"controller"**)  .setEntity(**"beans"**)  .setXml(**"mapper"**);   *//5. 整合配置* AutoGenerator ag = **new** AutoGenerator();   ag.setGlobalConfig(config)  .setDataSource(dsConfig)  .setStrategy(stConfig)  .setPackageInfo(pkConfig);   *//6. 执行* ag.execute();  } } |

Idea下载不了velocity 的jar 包，把仓库镜像暂时设置成下面的，用了改回原来。

|  |
| --- |
| <mirror>  <id>nexus</id>  <name>internal nexus repository</name>  <url>http://repo.maven.apache.org/maven2</url>  <mirrorOf>central</mirrorOf>  </mirror> |

## 6.3 ServiceImpl 说明

EmployeeServiceImpl 继承了 ServiceImpl 类,mybatis-plus 通过这种方

式为我们注入了 EmployeeMapper,这样可以使用 service 层默认为我们提供的很

多方法,也可以调用我们自己在 dao 层编写的操作数据库的方法.

|  |
| --- |
| **package** xiawenjie.mp.service.impl;  **import** xiawenjie.mp.beans.Employee; **import** xiawenjie.mp.mapper.EmployeeMapper; **import** xiawenjie.mp.service.EmployeeService; **import** com.baomidou.mybatisplus.service.impl.ServiceImpl; **import** org.springframework.stereotype.Service;  */\*\*  \* <p>  \* 服务实现类  \* </p>  \*  \** ***@author*** *xiawenjie  \** ***@since*** *2019-10-16  \*/* @Service **public class** EmployeeServiceImpl **extends** ServiceImpl<EmployeeMapper, Employee> **implements** EmployeeService {  } |

# 第7章：插件扩展

## 7.1 Mybatis 插件机制简介

### 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

### 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.

### 7.2 分页插件

物理分页

1. 插件位置： com.baomidou.mybatisplus.plugins.PaginationInterceptor

复制项目，把代码生成器的文件放到相应的位置

注册插件第一种方式，一般不这么用：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**>  *<!--注册插件，一帮不这么用-->  <!--<plugins>  <plugin interceptor="com.baomidou.mybatisplus.plugins.PaginationInterceptor"></plugin>  </plugins>-->* </**configuration**> |

第二种：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"> -->* <**bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!-- 别名处理 -->* <**property name="typeAliasesPackage" value="xia.wenjie.po"**></**property**>  *<!--注入全局配置-->* <**property name="globalConfig" ref="globalConfiguration"**></**property**>   *<!--插件注册-->  <!--plugins是数组-->* <**property name="plugins"**>  <**list**>  *<!-- 注册分页插件 -->* <**bean class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"**></**bean**>  </**list**>  </**property**>    </**bean**>   *<!--mybatiplus 的全局策略配置-->* <**bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration"**>  *<!--2.3版本以后dbColumnUnderline 默认值是true  如果不想开启，设置为false，那么实体就得在字段加 @TableField(value = "last\_name")  -->  <!--<property name="dbColumnUnderline" value="false"></property>-->    <!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**>  *<!--全局的主键-->* <**property name="idType" value="0"**></**property**>   </**bean**>    *<!--  配置 mybatis 扫描 mapper 接口的路径  -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**>  </**bean**> </**beans**> |

测试：

|  |
| --- |
| **package** xia.wenjie.test;  **import** java.sql.Connection; **import** java.util.List; **import** javax.sql.DataSource;  **import** com.baomidou.mybatisplus.mapper.EntityWrapper; **import** com.baomidou.mybatisplus.plugins.Page; **import** com.baomidou.mybatisplus.plugins.pagination.optimize.JsqlParserCountOptimize; **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** xia.wenjie.mapper.EmployeeMapper; **import** xia.wenjie.po.Employee;   **public class** DemoMybatisPlus {  ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);   @Test  **public void** testDemo() {  EmployeeMapper employeeMapper = **iocContext**.getBean(**"employeeMapper"**, EmployeeMapper.**class**);  *//分页插件测试 物理分  // SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee  // SELECT COUNT(1) FROM tbl\_employee  // SELECT id,last\_name AS lastName,email,gender,age FROM tbl\_employee LIMIT 3,3* List<Employee> employees = employeeMapper.selectPage(**new** Page<Employee>(2, 3), **null**);  System.***out***.println(employees);  } } |

再次运行不打印sql需要在 EmployeeMapper.xml

关闭二级缓存

控制器需要加入下面的依赖

|  |
| --- |
| *<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  <**version**>2.1.3.RELEASE</**version**> </**dependency**> |

注意：内存分页数据大的时候，频繁操作数据会不准，用物理分页靠谱

#### 注册分页插件后Page对象的使用

|  |
| --- |
| @Test **public void** testDemo() {  EmployeeMapper employeeMapper = **iocContext**.getBean(**"employeeMapper"**, EmployeeMapper.**class**);  Page<Employee> page = **new** Page<Employee>(2, 3);  List<Employee> employees = employeeMapper.selectPage(page, **null**);  System.***out***.println(**"==========获取分页相关的信息=========="**);   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(**"每页显示的条数"** + page.getSize());  System.***out***.println(**"是否有上一页"** + page.hasPrevious());  System.***out***.println(**"是否有上一页"** + page.hasNext());   *//将查询的结果封装到page对象* page.setRecords(employees); } |

### 7.3 执行分析插件

1)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。

|  |
| --- |
| *<!--插件注册--> <!--plugins是数组-->* <**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"**>  <**property name="stopProceed" value="true"**></**property**>  </**bean**>  </**list**> </**property**> |

|  |
| --- |
| **public class** DemoMybatisPlus {  ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);   EmployeeMapper **employeeMapper** = **iocContext**.getBean(**"employeeMapper"**, EmployeeMapper.**class**);   *//测试执行分析插件* @Test  **public void** deleteAlltestDemo() {  Integer integer = **employeeMapper**.delete(**null**);*//删除所有* } } |

|  |
| --- |
| Cause: com.baomidou.mybatisplus.exceptions.MybatisPlusException: com.baomidou.mybatisplus.exceptions.MybatisPlusException: Error: Full table operation is prohibited. SQL: DELETE FROM tbl\_employee |

### 7.4 性能分析插件

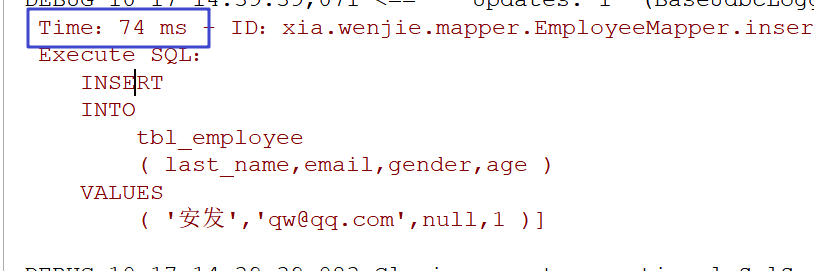
1)com.baomidou.mybatisplus.plugins.PerformanceInterceptor

2)性能分析拦截器，用于输出每条 SQL 语句及其执行时间

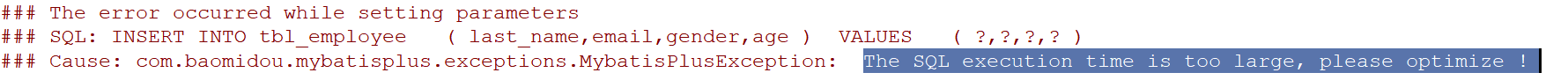
SQL 性能执行分析,开发环境使用，超过指定时间，停止运行。有助于发现问题

|  |
| --- |
| *<!-- SQL 执行性能分析，开发环境使用，线上不推荐。 maxTime 指的是 sql 最大执行时长 -->* <**bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor"**>  *<!--超过指定时间，停止运行-->  <!-- <property name="maxTime" value=""/>-->*  *<!--SQL是否格式化 默认false-->* <**property name="format" value="true"**/> </**bean**> |

|  |
| --- |
| *//性能分析插件* @Test **public void** inserttestDemo() {  Employee employee = **new** Employee();  employee.setLastName(**"安发"**);  employee.setAge(1);  employee.setEmail(**"qw@qq.com"**);  **employeeMapper**.insertAllColumn(employee); } |



再把超时指定时间打开，设置一个小于这个时间，测试



### 7.5 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

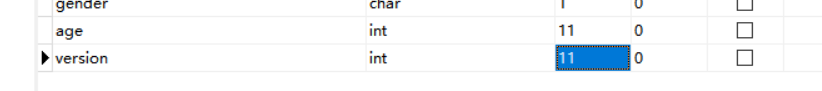
执行更新时， set version = yourVersion+1 where version = yourVersion 如果 version 不对，就更新失败

4)@Version 用于注解实体字段，必须要有。

|  |
| --- |
| *<!--注册乐观锁插件-->* <**bean class="com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor"**></**bean**> |

|  |
| --- |
| *//实体里面加上版本属性，getset方法，@Version注解* @Version **private** Integer **version**; **public** Integer getVersion() {  **return version**; } **public void** setVersion(Integer version) {  **this**.**version** = version; } |

该表加上version 字段。 手动给个值1

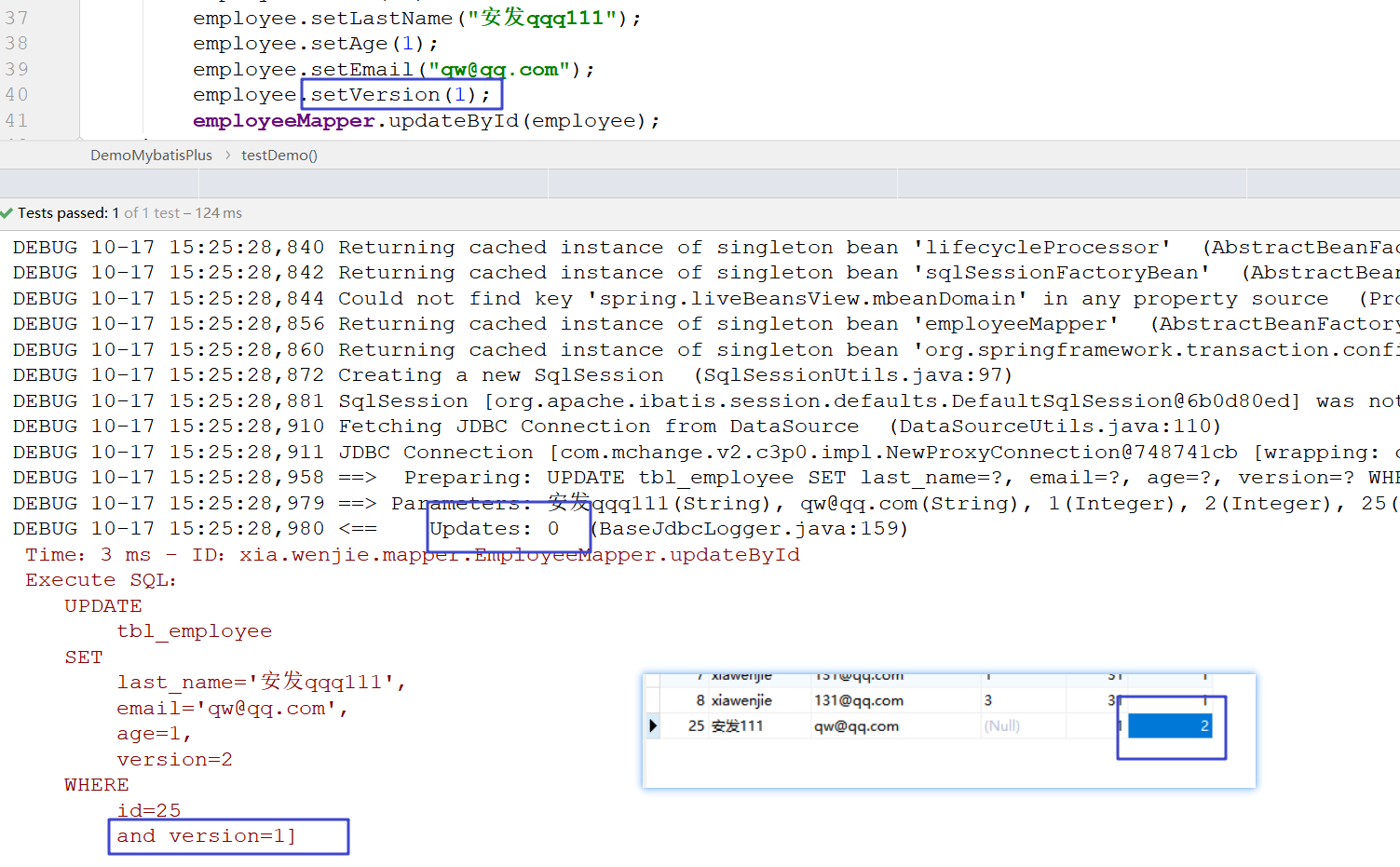


测试

|  |
| --- |
| *//测试乐观锁* @Test **public void** testDemo() {  Employee employee = **new** Employee();  employee.setId(25);  employee.setLastName(**"安发111"**);  employee.setAge(1);  employee.setEmail(**"qw@qq.com"**);  employee.setVersion(1);  **employeeMapper**.updateById(employee); } |



再运行测试，版本对应不上



# 第8章：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。

## 8.1 AutoSqlInjector

### 1)在 Mapper 接口中定义相关的 CRUD 方法

以删除全部为例演示自定义全局操作

|  |
| --- |
| **package** xia.wenjie.mapper; **import** com.baomidou.mybatisplus.mapper.BaseMapper; **import** xia.wenjie.po.Employee; **public interface** EmployeeMapper **extends** BaseMapper<Employee> {  **int** deleteAll(); } |

### 2)扩展 AutoSqlInjector inject 方法，实现 Mapper 接口中方法要注入的 SQL

|  |
| --- |
| **package** xia.wenjie.inject;  **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;  **import** java.util.Map;   */\*\*  \** ***@描述:*** *自定义全局操作  \** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* **public class** MyAutoSqlInjector **extends** AutoSqlInjector {  @Override  **public void** inject(Configuration configuration, MapperBuilderAssistant builderAssistant, Class<?> mapperClass, Class<?> modelClass, TableInfo table) {  *//将EmployeeMapper中定义的deleteAll， 处理成对应的MappedStatement对象，加入到configuration对象中。   //注入的SQL语句* String sql = **"delete from "** + table.getTableName();  *//注入的方法名 一定要与EmployeeMapper接口中的方法名一致* String method = **"deleteAll"**;  *//构造SqlSource对象* SqlSource sqlSource = **languageDriver**.createSqlSource(configuration, sql, modelClass);  *//构造一个删除的MappedStatement* **this**.addDeleteMappedStatement(mapperClass, method, sqlSource);  } } |

### 3)在 MP 全局策略中，配置 自定义注入器

|  |
| --- |
| *<?***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"> -->* <**bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!-- 别名处理 -->* <**property name="typeAliasesPackage" value="xia.wenjie.po"**></**property**>  *<!--注入全局配置-->* <**property name="globalConfig" ref="globalConfiguration"**></**property**>   *<!--插件注册-->  <!--plugins是数组-->* <**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">  <property name="stopProceed" value="true"></property>  </bean>-->   <!-- SQL 执行性能分析，开发环境使用，线上不推荐。 maxTime 指的是 sql 最大执行时长 -->* <**bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor"**>  *<!--超过指定时间，停止运行-->* <**property name="maxTime" value="1000"**/>  *<!--SQL是否格式化 默认false-->* <**property name="format" value="true"**/>  </**bean**>   *<!--注册乐观锁插件-->* <**bean class="com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor"**></**bean**>  </**list**>  </**property**>   </**bean**>    *<!--mybatiplus 的全局策略配置-->* <**bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration"**>  *<!--2.3版本以后dbColumnUnderline 默认值是true  如果不想开启，设置为false，那么实体就得在字段加 @TableField(value = "last\_name")  -->  <!--<property name="dbColumnUnderline" value="false"></property>-->    <!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**>  *<!--全局的主键-->* <**property name="idType" value="0"**></**property**>   *<!--注入自定义全局操作-->* <**property name="sqlInjector" ref="myAutoSqlInjector"**></**property**>   </**bean**>    *<!--自定注入器-->* <**bean id="myAutoSqlInjector" class="xia.wenjie.inject.MyAutoSqlInjector"**></**bean**>    *<!--  配置 mybatis 扫描 mapper 接口的路径  -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**>  </**bean**> </**beans**> |

测试：

|  |
| --- |
| **package** xia.wenjie.test;  **import** java.sql.Connection; **import** java.util.List; **import** javax.sql.DataSource;  **import** com.baomidou.mybatisplus.enums.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.DbType; **import** com.baomidou.mybatisplus.generator.config.rules.NamingStrategy; **import** com.baomidou.mybatisplus.mapper.EntityWrapper; **import** com.baomidou.mybatisplus.plugins.Page; **import** com.baomidou.mybatisplus.plugins.pagination.optimize.JsqlParserCountOptimize; **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** xia.wenjie.mapper.EmployeeMapper; **import** xia.wenjie.po.Employee;   **public class** DemoMybatisPlus {  ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);   EmployeeMapper **employeeMapper** = **iocContext**.getBean(**"employeeMapper"**, EmployeeMapper.**class**);  *//测试自定义注入器* @Test  **public void** testDemo() {  **employeeMapper**.deleteAll();  } } |

## 8.2 自定义注入器的应用之 逻辑删除

假删除、逻辑删除: 并不会真正的从数据库中将数据删除掉，而是将当前被删除的这条数据中的一个逻辑删除字段置为删除状态.

表tbl\_user 字段logic\_flag = 1 → -1

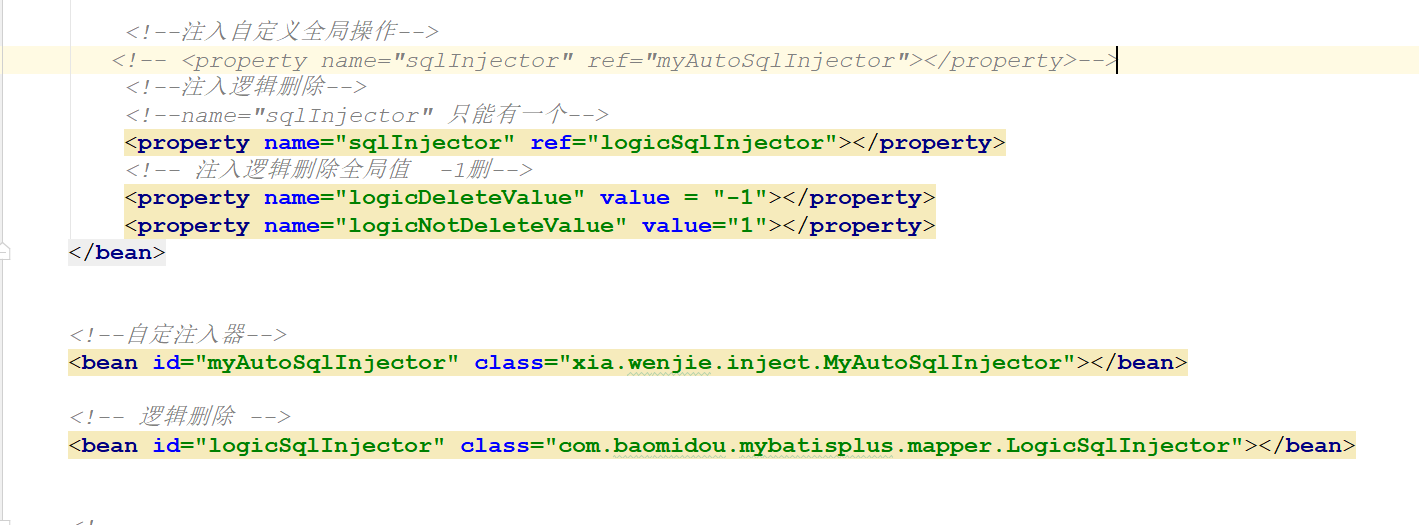
|  |
| --- |
| SET FOREIGN\_KEY\_CHECKS=0;  -- ----------------------------  -- Table structure for tbl\_user  -- ----------------------------  DROP TABLE IF EXISTS `tbl\_user`;  CREATE TABLE `tbl\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(20) DEFAULT NULL,  `logic\_flag` int(1) NOT NULL DEFAULT '1' COMMENT '软删除的 -1删',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;  -- ----------------------------  -- Records of tbl\_user  -- ----------------------------  INSERT INTO `tbl\_user` VALUES ('1', '张三', '1');  INSERT INTO `tbl\_user` VALUES ('2', '李四', '1');  INSERT INTO `tbl\_user` VALUES ('3', '王五', '1');  INSERT INTO `tbl\_user` VALUES ('4', '信息', '1');  INSERT INTO `tbl\_user` VALUES ('5', '得到', '1'); |

|  |
| --- |
| **package** xia.wenjie.po;  **import** com.baomidou.mybatisplus.annotations.TableLogic;  */\*\*  \** ***@描述:*** *\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* **public class** User {  **private** Integer **id**; *// `id` int(11) NOT NULL AUTO\_INCREMENT,* **private** String **name**; *// `name` varchar(20) DEFAULT NULL,* @TableLogic  **private** Integer **logicFlag**; *//`logic\_flag` int(1) NOT NULL DEFAULT '1' COMMENT '软删除的 -1删',* **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public** Integer getLogicFlag() {  **return logicFlag**;  }   **public void** setLogicFlag(Integer logicFlag) {  **this**.**logicFlag** = logicFlag;  }   @Override  **public** String toString() {  **return "User{"** +  **"id="** + **id** +  **", name='"** + **name** + **'\''** +  **", logicFlag="** + **logicFlag** +  **'}'**;  } } |

### 1)com.baomidou.mybatisplus.mapper.LogicSqlInjector

### 2)logicDeleteValue 逻辑删除全局值

### 3)logicNotDeleteValue 逻辑未删除全局值



|  |
| --- |
| *<!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**>  *<!--全局的主键-->* <**property name="idType" value="0"**></**property**>   *<!--注入自定义全局操作-->  <!-- <property name="sqlInjector" ref="myAutoSqlInjector"></property>-->  <!--注入逻辑删除-->  <!--name="sqlInjector" 只能有一个-->* <**property name="sqlInjector" ref="logicSqlInjector"**></**property**>  *<!-- 注入逻辑删除全局值 -1删-->* <**property name="logicDeleteValue" value = "-1"**></**property**>  <**property name="logicNotDeleteValue" value="1"**></**property**> </**bean**>   *<!--自定注入器-->* <**bean id="myAutoSqlInjector" class="xia.wenjie.inject.MyAutoSqlInjector"**></**bean**>  *<!-- 逻辑删除 -->* <**bean id="logicSqlInjector" class="com.baomidou.mybatisplus.mapper.LogicSqlInjector"**></**bean**>   *<!-- 配置 mybatis 扫描 mapper 接口的路径 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**> </**bean**> |

### 4)在 POJO 的逻辑删除字段 添加 @TableLogic 注解

|  |
| --- |
| **package** xia.wenjie.po;  **import** com.baomidou.mybatisplus.annotations.TableLogic;  */\*\*  \** ***@描述:*** *\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* **public class** User {  **private** Integer **id**; *// `id` int(11) NOT NULL AUTO\_INCREMENT,* **private** String **name**; *// `name` varchar(20) DEFAULT NULL,* @TableLogic  **private** Integer **logicFlag**; *//`logic\_flag` int(1) NOT NULL DEFAULT '1' COMMENT '软删除的 -1删',* **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public** Integer getLogicFlag() {  **return logicFlag**;  }   **public void** setLogicFlag(Integer logicFlag) {  **this**.**logicFlag** = logicFlag;  }   @Override  **public** String toString() {  **return "User{"** +  **"id="** + **id** +  **", name='"** + **name** + **'\''** +  **", logicFlag="** + **logicFlag** +  **'}'**;  } } |

mapper接口

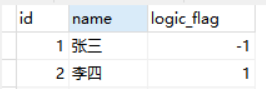
|  |
| --- |
| **package** xia.wenjie.mapper;  **import** com.baomidou.mybatisplus.mapper.BaseMapper; **import** xia.wenjie.po.User;  */\*\*  \** ***@描述:*** *\** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* **public interface** UserMapper **extends** BaseMapper<User> { } |

5)会在 mp 自带查询和更新方法的 sql 后面，追加『逻辑删除字段』=『LogicNotDeleteValue

默认值』 删除方法: deleteById()和其他 delete 方法, 底层 SQL 调用的是 update tbl\_xxx set 『逻辑删除字段』=『logicDeleteValue 默认值』

测试：

|  |
| --- |
| **package** xia.wenjie.test;   **import** org.junit.Test; **import** org.springframework.context.ApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** xia.wenjie.mapper.UserMapper;   **public class** DemoMybatisPlus {  ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);  UserMapper **userMapper** = **iocContext**.getBean(**"userMapper"**, UserMapper.**class**);   *//测试 软删除逻辑* @Test  **public void** deleteDemo() { *// UPDATE tbl\_user SET logic\_flag=-1 WHERE id=?* **userMapper**.deleteById(1);  } } |



再查找一下

|  |
| --- |
| *//测试 软删除逻辑* @Test  **public void** deleteDemo() { *// UPDATE tbl\_user SET logic\_flag=-1 WHERE id=? // userMapper.deleteById(1);   //SELECT id,`name`,logic\_flag AS logicFlag FROM tbl\_user WHERE id=? AND logic\_flag=1* **userMapper**.selectById(1);  } |

# 第 9 章：公共字段自动填充

## 9.1 元数据处理器接口

com.baomidou.mybatisplus.mapper.MetaObjectHandler

如果在更新或插入操作时可以给属性预设一个固定的值。

insertFill(MetaObject metaObject)

updateFill(MetaObject metaObject)

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. 支持对 Object 、Map、Collection

等对象进行包装

本质上 metaObject 获取对象的属性值或者是给对象的属性设置值，最终是要

通过 Reflector 获取到属性的对应方法的 Invoker, 最终 invoke.

## 9.2 开发步骤

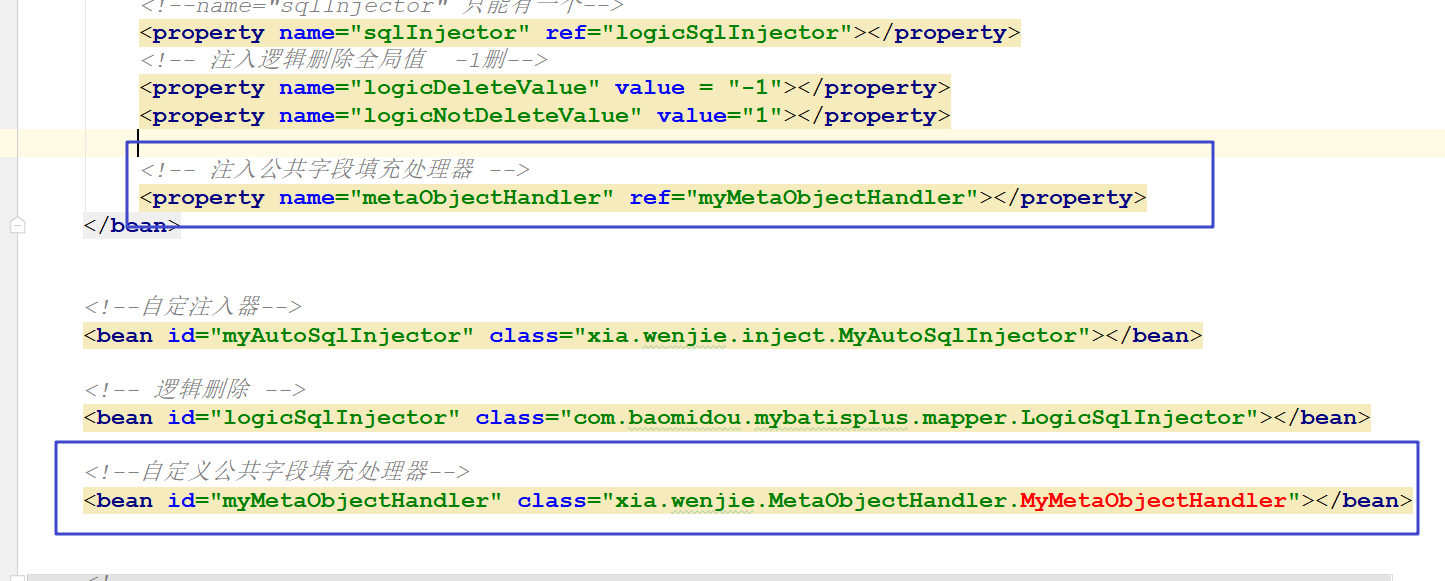
### 1)注解填充字段 @TableFile(fill = FieldFill.INSERT) 查看 FieldFill

|  |
| --- |
| **public class** User {  **private** Integer **id**;  @TableField(fill = FieldFill.***INSERT\_UPDATE***) *//插入和更新填充字段* **private** String **name**;  @TableLogic  **private** Integer **logicFlag**; *//软删除的 -1删',* **public** Integer getId() {  **return id**;  } |

### 2)自定义公共字段填充处理器

|  |
| --- |
| **package** xia.wenjie.MetaObjectHandler;  **import** com.baomidou.mybatisplus.mapper.MetaObjectHandler; **import** org.apache.ibatis.reflection.MetaObject;  */\*\*  \** ***@描述:*** *自定义公共字段填充处理器  \** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* **public class** MyMetaObjectHandler **extends** MetaObjectHandler {  */\*\*  \* 插入操作 自动填充  \*/* **public void** insertFill(MetaObject metaObject) {  *//获取到需要被填充的字段的值  //pojo 的属性名* Object fieldName = getFieldValByName(**"name"**, metaObject);  **if** (fieldName==**null**) {  setFieldValByName(**"name"**,**"默认名字：孙悟空"**, metaObject);  }  }  */\*\*  \* 修改操作 自动填充  \*/* **public void** updateFill(MetaObject metaObject) {  *//获取到需要被填充的字段的值  //pojo 的属性名* Object fieldName = getFieldValByName(**"name"**, metaObject);  **if** (fieldName==**null**) {  setFieldValByName(**"name"**,**"默认名字：猪八戒"**, metaObject);  }  } } |

### 3) MP 全局注入 自定义公共字段填充处理器



|  |
| --- |
| <**property name="logicDeleteValue" value = "-1"**></**property**>  <**property name="logicNotDeleteValue" value="1"**></**property**>    *<!-- 注入公共字段填充处理器 -->* <**property name="metaObjectHandler" ref="myMetaObjectHandler"**></**property**> </**bean**>   *<!--自定注入器-->* <**bean id="myAutoSqlInjector" class="xia.wenjie.inject.MyAutoSqlInjector"**></**bean**>  *<!-- 逻辑删除 -->* <**bean id="logicSqlInjector" class="com.baomidou.mybatisplus.mapper.LogicSqlInjector"**></**bean**>  *<!--自定义公共字段填充处理器-->* <**bean id="myMetaObjectHandler" class="xia.wenjie.MetaObjectHandler.MyMetaObjectHandler"**></**bean**> |

测试：不设置neme属性的值

|  |
| --- |
| *//测试 字段填充器* @Test **public void** testDemo() {  User user = **new** User();  user.setLogicFlag(2);  *// INSERT INTO tbl\_user( `name`, logic\_flag ) VALUES( '默认名字：猪八戒', 2 )* **userMapper**.insert(user); }  *//测试 字段填充器* @Test **public void** testDemo() {  User user = **new** User();  user.setName(**"xiaxia"**);  user.setLogicFlag(1); *// INSERT INTO tbl\_user( `name`, logic\_flag ) VALUES( 'xiaxia', 1 )* **userMapper**.insert(user); }  *//测试 字段填充器* @Test  **public void** testDemo() {  User user = **new** User();  user.setId(6); *// UPDATE tbl\_user SET `name`='默认名字：孙悟空' WHERE id=6 AND logic\_flag=1* **userMapper**.updateById(user);  } |

# 第10章Oracle主键Sequence

MySQL: 支持主键自增。 IdType.Auto

Oracle: 序列(Sequence)

1.复制ojdbc14驱动包到仓库com目录下，加入依赖

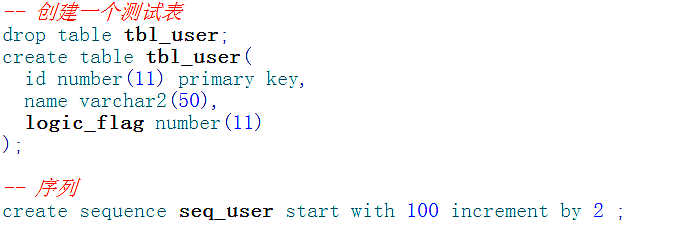
|  |
| --- |
| *<!-- Oracle驱动: 因为Oracle授权的问题，不能从Maven的仓库中下载到Oracle驱动. -->* <**dependency**>  <**groupId**>com.oracle</**groupId**>  <**artifactId**>ojdbc14</**artifactId**>  <**version**>10.2.0.4.0</**version**> </**dependency**> |

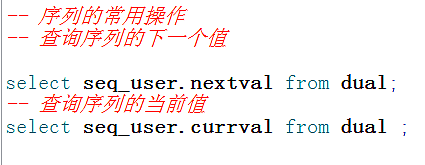
1. ad.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mp  jdbc.username=root  jdbc.password=1234  orcl.driver=oracle.jdbc.OracleDriver  orcl.url=jdbc:oracle:thin:@localhost:1521:xe  orcl.username=system  orcl.password=1234 |

3.数据源配置 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"**/>   *<!--mysql-->  <!--<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>-->   <!-- Oracle -->* <**bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>  <**property name="driverClass" value="${orcl.driver}"**></**property**>  <**property name="jdbcUrl" value="${orcl.url}"**></**property**>  <**property name="user" value="${orcl.username}"**></**property**>  <**property name="password" value="${orcl.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"> -->* <**bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!-- 别名处理 -->* <**property name="typeAliasesPackage" value="xia.wenjie.po"**></**property**>  *<!--注入全局配置-->* <**property name="globalConfig" ref="globalConfiguration"**></**property**>   *<!--插件注册-->  <!--plugins是数组-->* <**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">  <property name="stopProceed" value="true"></property>  </bean>-->   <!-- SQL 执行性能分析，开发环境使用，线上不推荐。 maxTime 指的是 sql 最大执行时长 -->* <**bean class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor"**>  *<!--超过指定时间，停止运行-->* <**property name="maxTime" value="1000"**/>  *<!--SQL是否格式化 默认false-->* <**property name="format" value="true"**/>  </**bean**>   *<!--注册乐观锁插件-->* <**bean class="com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor"**></**bean**>  </**list**>  </**property**>   </**bean**>    *<!--mybatiplus 的全局策略配置-->* <**bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration"**>  *<!--2.3版本以后dbColumnUnderline 默认值是true  如果不想开启，设置为false，那么实体就得在字段加 @TableField(value = "last\_name")  -->  <!--<property name="dbColumnUnderline" value="false"></property>-->    <!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**>  *<!--全局的主键-->* <**property name="idType" value="0"**></**property**>   *<!--注入自定义全局操作-->  <!-- <property name="sqlInjector" ref="myAutoSqlInjector"></property>-->  <!--注入逻辑删除-->  <!--name="sqlInjector" 只能有一个-->* <**property name="sqlInjector" ref="logicSqlInjector"**></**property**>  *<!-- 注入逻辑删除全局值 -1删-->* <**property name="logicDeleteValue" value = "-1"**></**property**>  <**property name="logicNotDeleteValue" value="1"**></**property**>   *<!-- 注入公共字段填充处理器 -->* <**property name="metaObjectHandler" ref="myMetaObjectHandler"**></**property**>  </**bean**>    *<!--自定注入器-->* <**bean id="myAutoSqlInjector" class="xia.wenjie.inject.MyAutoSqlInjector"**></**bean**>   *<!-- 逻辑删除 -->* <**bean id="logicSqlInjector" class="com.baomidou.mybatisplus.mapper.LogicSqlInjector"**></**bean**>   *<!--自定义公共字段填充处理器-->* <**bean id="myMetaObjectHandler" class="xia.wenjie.MetaObjectHandler.MyMetaObjectHandler"**></**bean**>    *<!--  配置 mybatis 扫描 mapper 接口的路径  -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**>  </**bean**> </**beans**> |





## 1) 实体类配置主键 Sequence @KeySequence(value=”序列名”，clazz=xxx.class 主键属性类

## 型)

|  |
| --- |
| */\*\*  \** ***@描述:*** *oracle表  \** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/17  \*/* @KeySequence(value=**"seq\_user"**,clazz=Integer.**class**)**public class** User {  *//@TableId(type=IdType.INPUT)* **private** Integer **id**;  @TableField(fill = FieldFill.***INSERT\_UPDATE***) *//插入和更新填充字段* **private** String **name**;  @TableLogic  **private** Integer **logicFlag**; *//软删除的 -1删',* |

## 2) 全局 MP 主键生成策略为 IdType.INPUT

|  |
| --- |
| *<!--全局的表前缀 -->* <**property name="tablePrefix" value="tbl\_"**></**property**> *<!--全局的主键--> <!--mysql的 数据库ID自增 0， "用户输入ID" 1--> <!--<property name="idType" value="0"></property>--> <!--oracle-->* <**property name="idType" value="1"**></**property**> |

## 3) 全局 MP 中配置 Oracle 主键 Sequence

com.baomidou.mybatisplus.incrementer.OracleKeyGenerator

|  |
| --- |
| *<!-- 注入公共字段填充处理器 -->* <**property name="metaObjectHandler" ref="myMetaObjectHandler"**></**property**>   *<!-- 注入Oracle主键Sequence -->* <**property name="keyGenerator" ref="oracleKeyGenerator"**></**property**>  </**bean**>    *<!--自定注入器-->* <**bean id="myAutoSqlInjector" class="xia.wenjie.inject.MyAutoSqlInjector"**></**bean**>   *<!-- 逻辑删除 -->* <**bean id="logicSqlInjector" class="com.baomidou.mybatisplus.mapper.LogicSqlInjector"**></**bean**>   *<!--自定义公共字段填充处理器-->* <**bean id="myMetaObjectHandler" class="xia.wenjie.MetaObjectHandler.MyMetaObjectHandler"**></**bean**>   *<!-- 配置Oracle主键Sequence -->* <**bean id="oracleKeyGenerator" class="com.baomidou.mybatisplus.incrementer.OracleKeyGenerator"**></**bean**>  *<!--  配置 mybatis 扫描 mapper 接口的路径  -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="xia.wenjie.mapper"**></**property**>  </**bean**> </**beans**> |
| 测试  **public class** DemoMybatisPlus {  ApplicationContext **iocContext** = **new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);  UserMapper **userMapper** = **iocContext**.getBean(**"userMapper"**, UserMapper.**class**);  */\*\*  \* 测试Oracle 主键 Sequence  \*/* @Test  **public void** testOracle() {  User user = **new** User();  user.setLogicFlag(1);  user.setName(**"OracleSEQ"**);  **userMapper**.insert(user);  } } |

## 4) 可以将@keySequence 定义在父类中，可实现多个子类对应的多个表公用一个 Sequence

加入有很多个的实体，每个都要配置@KeySequence(value=**"seq\_user"**,clazz=Integer.**class**)比较麻烦，可以搞个父类配置，其他的类继承父类就行

|  |
| --- |
| **package** xia.wenjie.po; **import** com.baomidou.mybatisplus.annotations.KeySequence; */\*\*  \** ***@描述:*** *实体公共父类  \** ***@功能:*** *\** ***@作者:*** *xiawenjie  \** ***@时间:*** *2019/10/18  \*/* @KeySequence(value = **"seq\_user"**, clazz = Integer.**class**) **public abstract class** Parent { } |
| *//@KeySequence(value="seq\_user",clazz=Integer.class)* **public class** User **extends** Parent { |

# 第 11 章Idea快速开发插件

MybatisX 辅助 idea 快速开发插件，为效率而生.

可以实现 java 与 xml 相互的跳转，根据 Mapper 接口中的方法自动生成 xml 结构.

官方安装： File -> Settings -> Plugins -> Browse Repositories.. 输入 mybatisx 安装下载

Jar 安装： File -> Settings -> Plugins -> Install plugin from disk.. 选中 mybatisx..jar 安装

