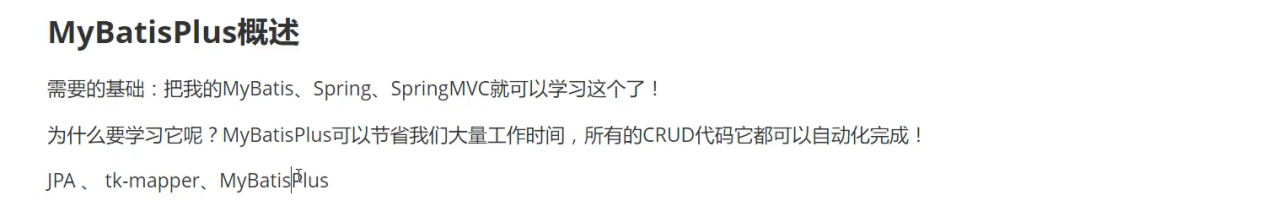
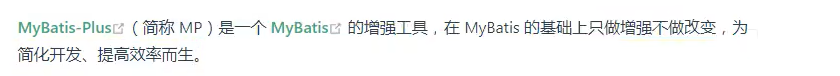
mybatis-plus：



是什么？

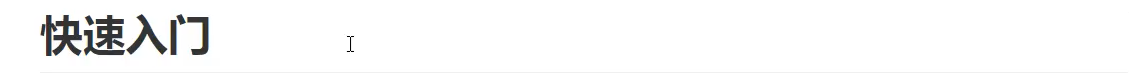




## **特性**

* ****无侵入****：只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑
* ****损耗小****：启动即会自动注入基本 CURD，性能基本无损耗，直接面向对象操作
* ****强大的 CRUD 操作****：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求
* ****支持 Lambda 形式调用****：通过 Lambda 表达式，方便的编写各类查询条件，无需再担心字段写错
* ****支持主键自动生成****：支持多达 4 种主键策略（内含分布式唯一 ID 生成器 - Sequence），可自由配置，完美解决主键问题
* ****支持 ActiveRecord 模式****：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可进行强大的 CRUD 操作
* ****支持自定义全局通用操作****：支持全局通用方法注入（ Write once, use anywhere ）
* ****内置代码生成器****：采用代码或者 Maven 插件可快速生成 Mapper 、 Model 、 Service 、 Controller 层代码，支持模板引擎，更有超多自定义配置等您来使用（自动帮你生成代码）
* ****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

快速入门：



地址：

<https://baomidou.com/guide/quick-start.html#%E5%88%9D%E5%A7%8B%E5%8C%96%E5%B7%A5%E7%A8%8B>

使用第三方组件︰

1、导入对应的依赖

2、研究依赖如何配置

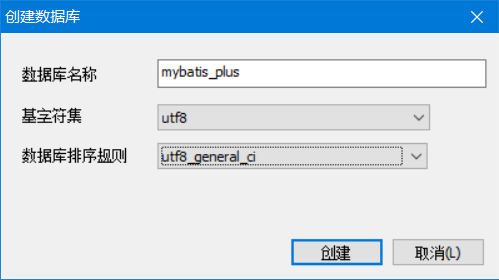
3、代码如何编写

4、提高扩展技术能力!

步骤：



1、创建数据库



2、创建表

DROP TABLE IF EXISTS user;

CREATE TABLE user(

id BIGINT(20) NOT NULL COMMENT '主键ID',

name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',

age INT(11) NULL DEFAULT NULL COMMENT '年龄',

email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY (id));

1. 插入表数据

DELETE FROM USER;

INSERT INTO USER (id, NAME, age, email) VALUES

(1, 'Jone', 18, 'test1@baomidou.com'),

(2, 'Jack', 20, 'test2@baomidou.com'),

(3, 'Tom', 28, 'test3@baomidou.com'),

(4, 'Sandy', 21, 'test4@baomidou.com'),

(5, 'Billie', 24, ['test5@baomidou.com');](mailto:'test5@baomidou.com');)

1. 导入依赖：只要导入mp就不要导入m了

<!-- 数据库jdbc依赖-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!-- mybatis\_plus依赖-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.0.5</version>

</dependency>



5、创建UserMapper，继承BaseMapper<T>然后加上mapper层的注解@Repository

@Repository  
public interface UserMapper extends BaseMapper<User> {  
  
}

6、在启动类上加上注解@MapperScan("com.zsp.mapper")

//扫描mybatis-plus接口  
@MapperScan("com.zsp.mapper")  
  
@SpringBootApplication  
public class MybatisPlusApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(MybatisPlusApplication.class, args);  
 }  
  
}

1. 在测试类里面使用

@SpringBootTest  
class MybatisPlusApplicationTests {  
  
 @Autowired  
 UserMapper userMapper;  
 @Test  
 void contextLoads() {  
 List<User> users = userMapper.selectList(null);  
 users.forEach(System.out::println);  
 }  
  
}

****

**配置日志：**



我们所有的sql现在是不可见的，我们希望知道它是怎么执行的，所以我们必须要看日志!在application.yml里面配置

mybatis-plus:  
 configuration:  
 log-impl: org.apache.ibatis.logging.stdout.StdOutImpl

CRUD扩展：





**主键生成策略：雪花算法**



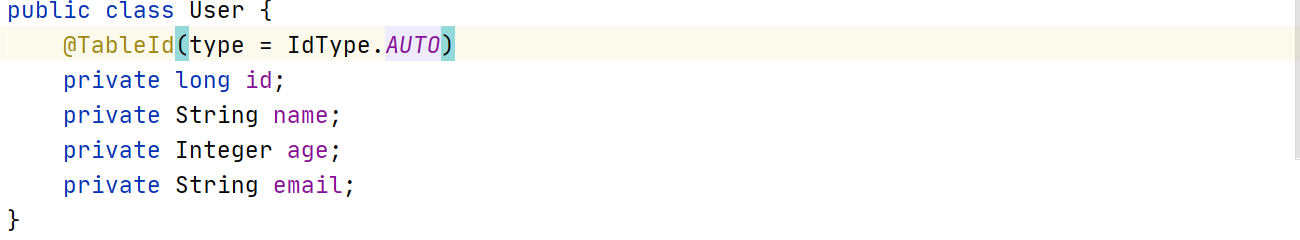
分布式系统唯─id生成：<https://www.cnblogs.com/haoxinyue/p/5208136.html>

snowflake是Twitter开源的分布式ID生成算法，结果是一个long型的ID。其核心思想是：使用41bit作为毫秒数，10bit作为机器的ID（5个bit是数据中心，5个bit的机器ID），12bit作为毫秒内的流水号（意味着每个节点在每毫秒可以产生 4096 个 ID），最后还有一个符号位，永远是0。具体实现的代码可以参看

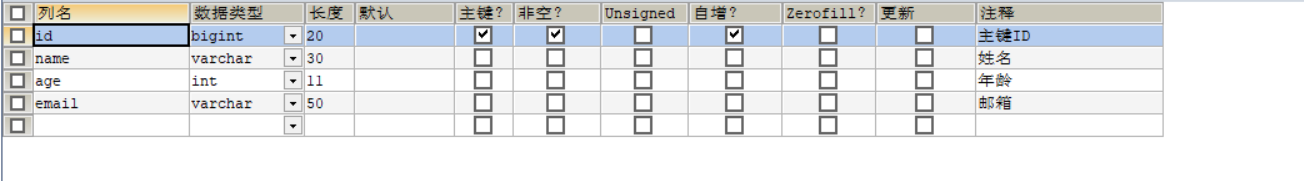
主键自增策略：@TableId(type=...)













其余的type： 

更新数据：使用mybatis-plus会自动使用动态sql，如下图，没有其他属性时



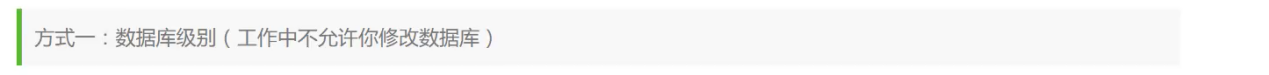
所有的sql都是动态的！

自动填充：



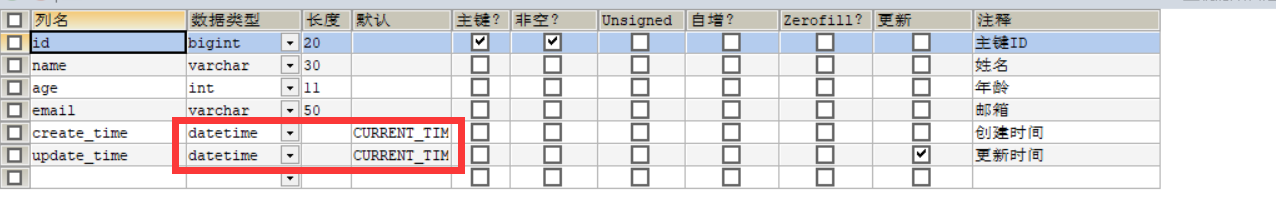


方式1：

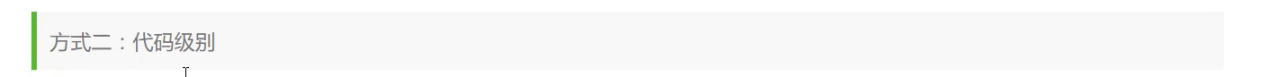




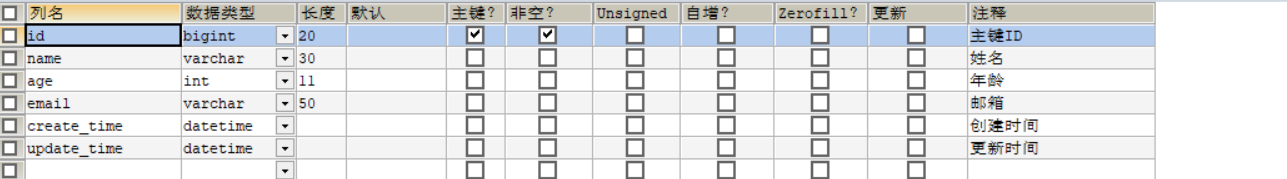
然后给上默认值CURRENT\_TIMESTAMP，并且更新时间给上更新的勾选。



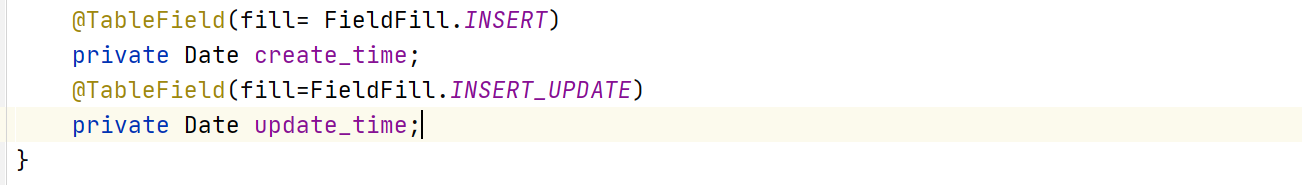
方式2：代码级别



1. 删除数据库的默认值、更新操作



2、在代码上给上注解：@TableField(fill=FieldFill.INSERT\_UPDATE)



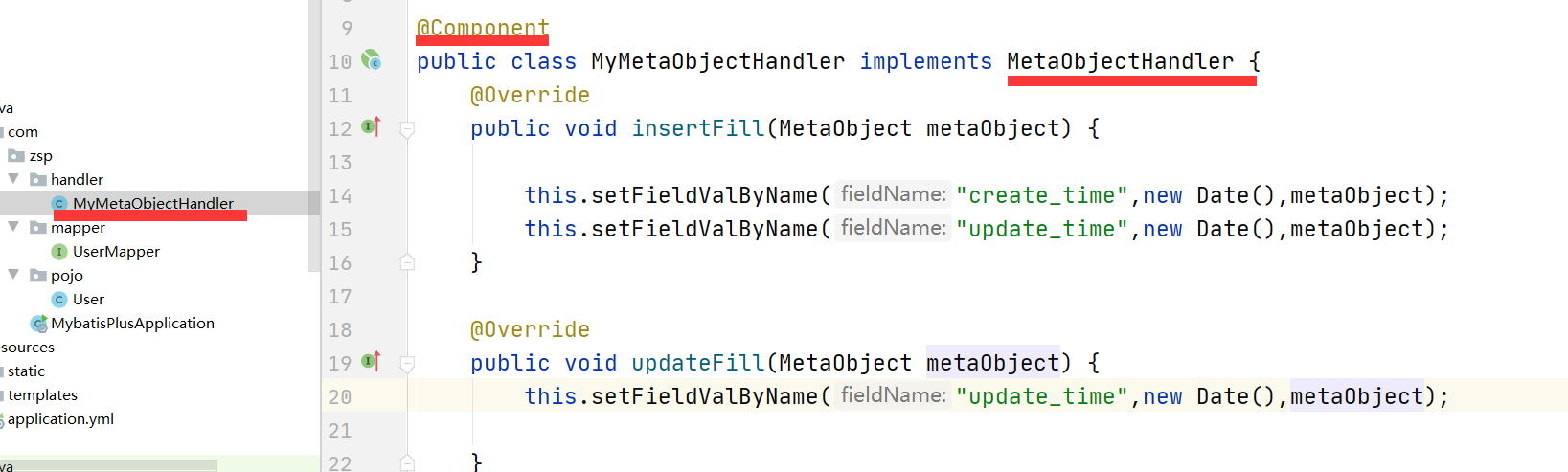
3、编写组件，给上注解实现MetaObjectHandler

insertFill

只对插入操作负责 type=FieldFill.INSERT

updateFill

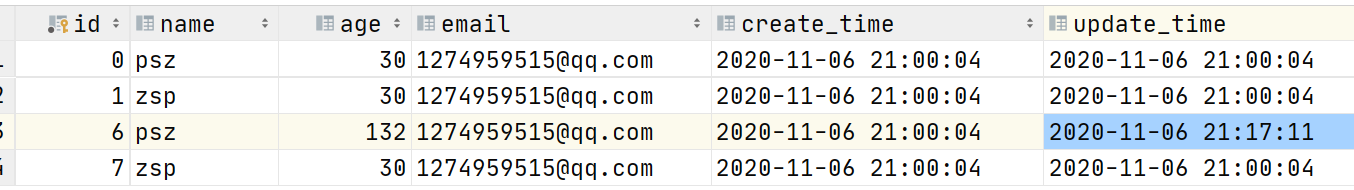
只对更新操作负责 type= FieldFill.INSERT\_UPDATE



切记一定要写上component还有就说filedName是pojo类里的属性名

setFieldValByName(String fieldName, Object fieldVal, MetaObject metaObject)  
//需要填入的三个数据分别是插数值名字，插入的数值，还有方法自带的metaObject

3、进行更新操作，发现已经实现：



**锁**

乐观锁:故名思意十分乐观，它总是认为不会出现问题，无论千什么不去上锁!如果出现了问题，再次更新值测试

悲观锁︰故名思意十分悲观，它总是认为总是出现问题，无论干什么都会上锁!再去操作!

**乐观锁**

## [#](https://baomidou.com/guide/interceptor-optimistic-locker.html" \l "optimisticlockerinnerinterceptor)**OptimisticLockerInnerInterceptor**

当要更新一条记录的时候，希望这条记录没有被别人更新乐观锁实现方式：

1取出记录时，获取当前version

2更新时，带上这个version

3执行更新时， set version = newVersion where version = oldVersion

4如果version不对，就更新失败

例如：

设定当前版本号为1，A程序执行的时候会带上版本号为1，然后再修改版本号为2，此时b也在进行修改也必须带上版本号1也修改成2，假定a先修改完把版本号改为了2，此时b的版本号和原版本号对应不上，b的操作就不能执行了。

乐观锁:1、A先查询，获得版本号version = 1，A先修改完毕版本号为2

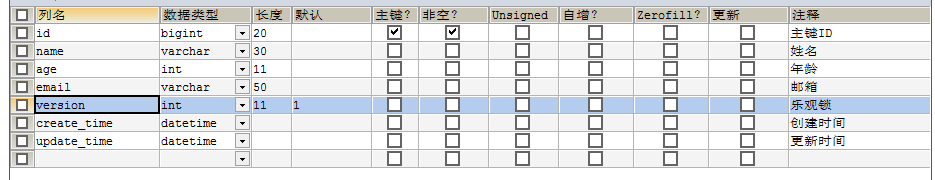
update user set name = "kuangshen"， version = version + 1where id= 2 and version = 1

2、B也进行修改，这个时候version = 1，但是当前版本为2了修改失败!

update user set name = "kuangshen "， version = version + 1where id = 2 and version = 1

测试乐观锁：

1在数据库添加字段：



2实体类添加对应字段，并且添加注解

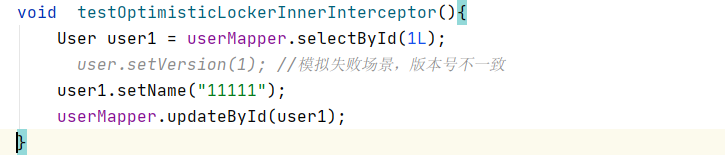
@Version  
private Integer version;

3写配置类：主启动类的扫描包直接移到配置类里

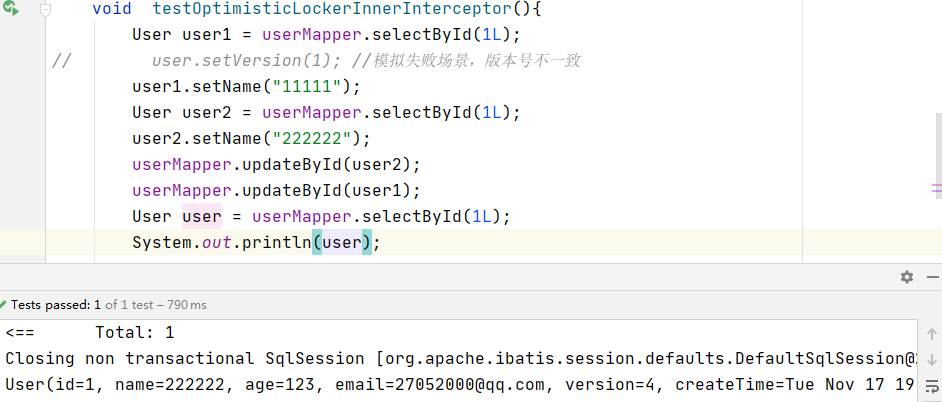


测试：

必须携带正确的版本号才可以进行操作：

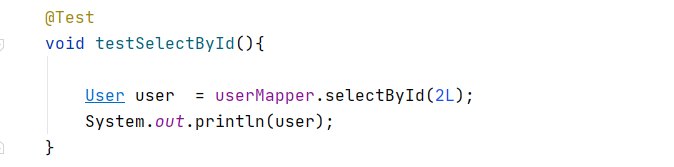


执行测试模拟并发的时候，会发现只有携带了正确版本号的才能执行，并不会被随意覆盖。如果要第二个能跑，需要使用自旋锁。

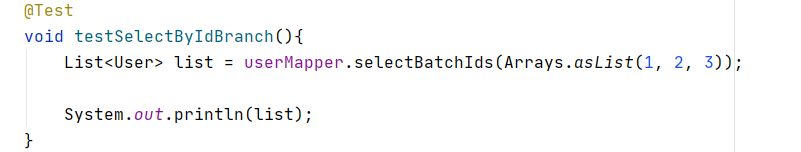


查询和分页查询：

普通的根据id查询：



多个id查询：



根据自己的条件查询：



**分页查询**

分页在网站使用的十分之多!

1. 原始的limit进行分页

2、pageHelper第三方插件

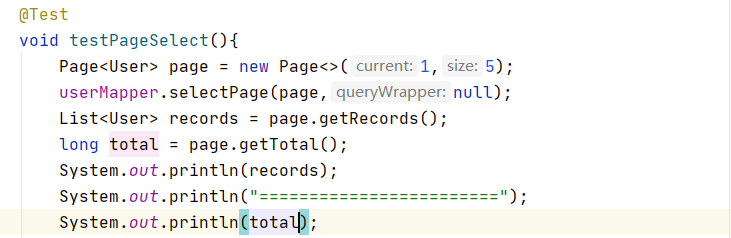
3、MP其实也内置了分页插件!

**如何使用**

1配置分页插件（这是新的分页配置类

@Bean  
public MybatisPlusInterceptor mybatisPlusInterceptor() {  
 MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();  
 interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));  
 return interceptor;  
}

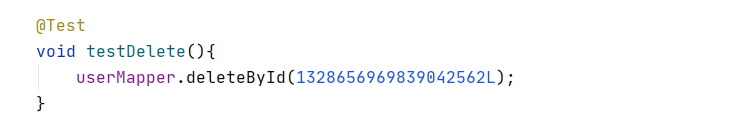
2使用Page对象即可



**删除操作：**

物理删除：从数据库直接删除

正常按id删除：



删除多位：

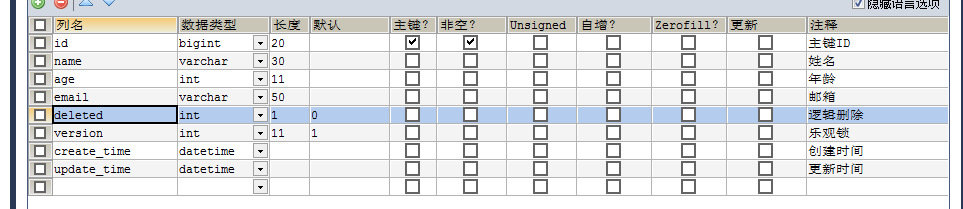


**逻辑删除：在数据库里没有删除而是设置了一个字段让他失效！**

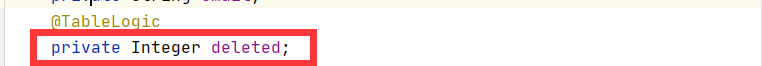
**deleted=0 =>deleted=1**

具体步骤：

1在数据库增加一个字段



2在pojo类里面增加deleted字段：打上@TableLogic

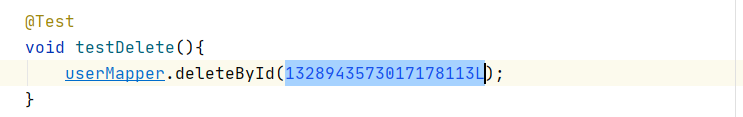


3配置配置类：新版本已经不需要了

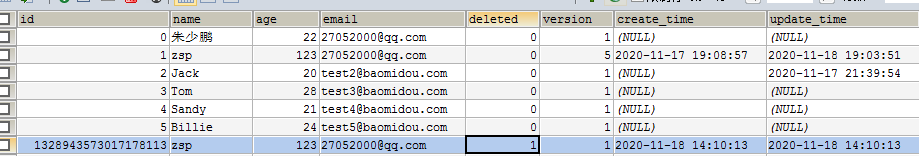
// 逻辑删除，新版本已经不需要配置这个  
// @Bean  
// public ISqlInjector sqlInjector() {  
// return new LogicSqlInjector();  
//  
// }

4测试

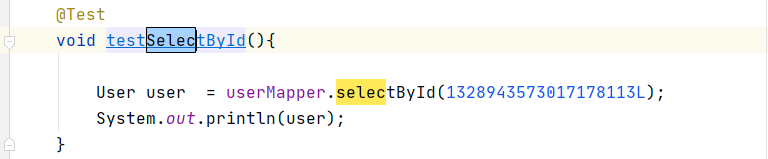
删除这条数据：

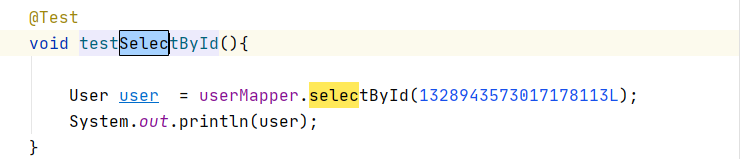


会发现数据还在，只是标为了删除：



并且也查询不到该数据





性能分析插件：

我们在平时的开发中，会遇到一些慢sql。测试! druid....

作用︰性能分析拦截器，用于输出每条SQL语句及其执行时间MP也提供性能分析插件，如果超过这个时间就停止运行!

1、导入插件

2、测试使用

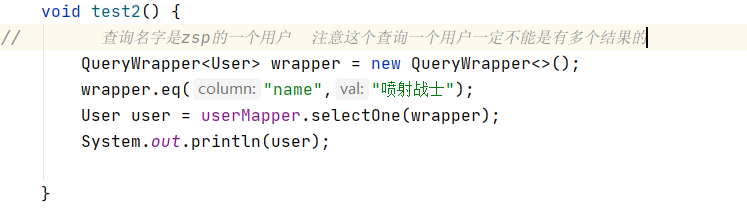
**条件构造器：**

十分重要：wrapper

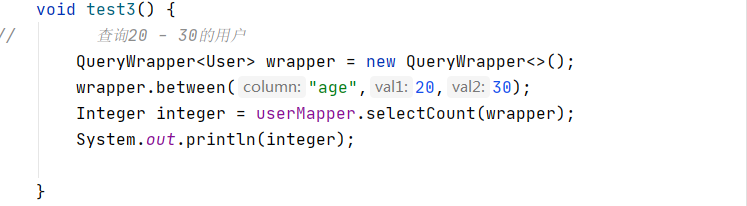
**开始使用：**

编写代码，使QueryWrappery：注意这个可以使用链式编程



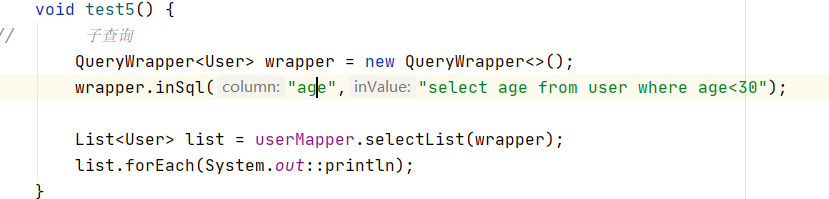


范围查询between：



模糊查询：like，左右查询  

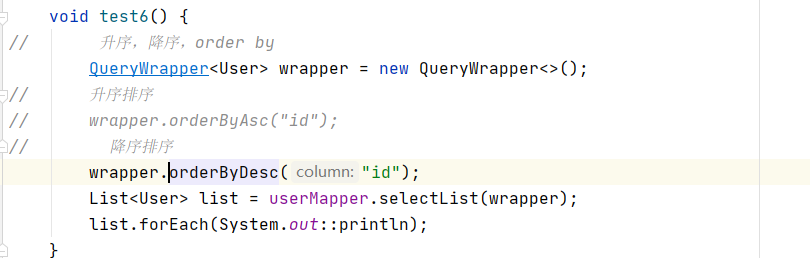

子查询：



查询的语句：

SELECT id,name,age,email,deleted,version,create\_time,update\_time FROM user WHERE deleted=0 AND (age IN (select age from user where age<30))

升序和降序排序：order。。。 by



**代码生成器:**

dao、pojo、service、controller都给我自己去编写完成!

AutoGenerator是MyBatis-Plus 的代码生成器，通过AutoGenerator可以快速生成Entity、Mapper、Mapper XML.、Service,Controller等各个模块的代码，极大的提升了开发效率。

使用代码生成器：

1导入依赖

<!-- 导入德鲁伊连接池和数据库连接-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.23</version>

</dependency>

<!-- MP-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.4.1</version>

</dependency>

<!-- MP的代码生成器-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>3.4.1</version>

</dependency>

<!-- MP的代码生成器依赖-->

<dependency>

<groupId>org.apache.velocity</groupId>

<artifactId>velocity-engine-core</artifactId>

<version>2.2</version>

</dependency>

导入配置类：

@MapperScan("com.zsp.mapper")  
@EnableTransactionManagement //开启事务  
@Configuration //配置类  
public class MybatisPlusConfig {  
// 性能分析插件  
// 逻辑删除，新版本已经不需要配置这个  
// @Bean  
// public ISqlInjector sqlInjector() {  
// return new LogicSqlInjector();  
//  
// }  
// mybatis乐观锁配置类  
 @Bean  
 public MybatisPlusInterceptor MybatisPlusInterceptor() {  
 MybatisPlusInterceptor mybatisPlusInterceptor = new MybatisPlusInterceptor();  
 mybatisPlusInterceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());  
 return mybatisPlusInterceptor;  
 }  
// mybatis的分页查询配置类  
 @Bean  
 public MybatisPlusInterceptor mybatisPlusInterceptor() {  
 MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();  
 interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));  
 return interceptor;  
 }  
  
  
}

写自动生成策略即可：

// 代码自动生成器对象  
 AutoGenerator autoGenerator = new AutoGenerator();  
// 配置策略  
  
// 1.全局配置 GlobalConfig  
 GlobalConfig gc = new GlobalConfig();  
// 获取当前项目目录  
 String projectPath = System.getProperty("user.dir");  
// 设置代码生成器自动生成路径  
 gc.setOutputDir(projectPath+"/src/main/java");  
// 设置作者名字  
 gc.setAuthor("zsp");  
// 自动生成后是否打开文件夹  
 gc.setOpen(false);  
// 是否覆盖原本的文件  
 gc.setFileOverride(false);  
// 去掉service的前缀  
 gc.setServiceName("%sService");  
// 设置日期类型  
 gc.setDateType(DateType.ONLY\_DATE);  
// 设置生成swagger  
// gc.setSwagger2(true);  
// 把全局配置丢进代码生成器中  
 autoGenerator.setGlobalConfig(gc);  
  
// 2.设置数据源 就是数据库配置  
 DataSourceConfig dataSourceConfig =new DataSourceConfig();  
 dataSourceConfig.setUrl("jdbc:mysql://localhost:3306/zspdisk?serverTimezone=Asia/Shanghai");  
 dataSourceConfig.setDbType(DbType.MYSQL);  
 dataSourceConfig.setDriverName("com.mysql.cj.jdbc.Driver");  
 dataSourceConfig.setUsername("root");  
 dataSourceConfig.setPassword("123456");  
 autoGenerator.setDataSource(dataSourceConfig);  
// 3.包的配置  
 PackageConfig pc = new PackageConfig();  
// 生成的模块名字  
// pc.setModuleName("user");  
// 生成在什么文件夹内  
 pc.setParent("com.zsp");  
// 各层的名字  
 pc.setEntity("entity");  
 pc.setMapper("mapper");  
 pc.setService("service");  
 pc.setController("controller");  
 autoGenerator.setPackageInfo(pc);  
// 4.策略：映射对应的表  
 StrategyConfig strategy = new StrategyConfig();  
 strategy.setInclude("user","user\_file","user\_folder","origin\_file"); //要映射的表名  
 strategy.setNaming(NamingStrategy.underline\_to\_camel); //开启包的命名规则,下划线转驼峰  
 strategy.setColumnNaming(NamingStrategy.underline\_to\_camel);//列也支持下划线转驼峰  
 strategy.setEntityLombokModel(true);//开启lombok  
 strategy.setLogicDeleteFieldName("status");  
// 自动填充策略:设置自动更新操作时间  
 TableFill userCreate = new TableFill("user\_create", FieldFill.INSERT);  
 TableFill userModified = new TableFill("user\_modified", FieldFill.INSERT\_UPDATE);  
// 把自动填充策略放进去  
 ArrayList<TableFill>tableFill = new ArrayList<>();  
 tableFill.add(userCreate);  
 tableFill.add(userModified);  
 strategy.setTableFillList(tableFill);  
// 乐观锁  
 strategy.setVersionFieldName("version");  
// 开启restful风格  
 strategy.setRestControllerStyle(true);  
// 开启url下划线命名类似于 localhost:8080/user\_1234\_2312  
 strategy.setControllerMappingHyphenStyle(true);  
// 放入策略  
 autoGenerator.setStrategy(strategy);  
 // 执行代码生成器  
 autoGenerator.execute();  
 }