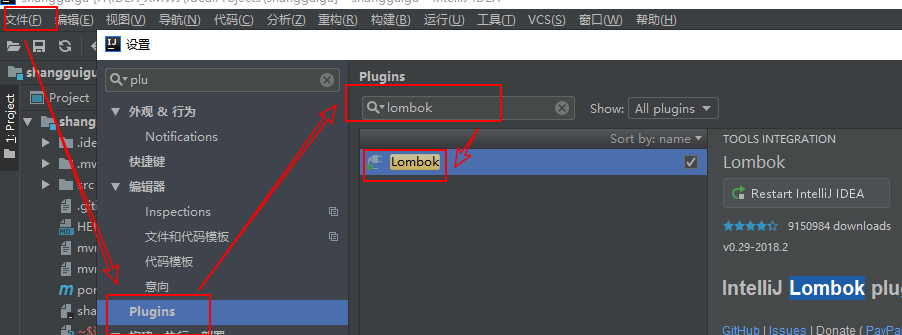
尚硅谷1——谷粒学院

# 知识点零碎总结：

Idea安装插件，实例lombok插件，在pom.xml文件里面添加插件引用

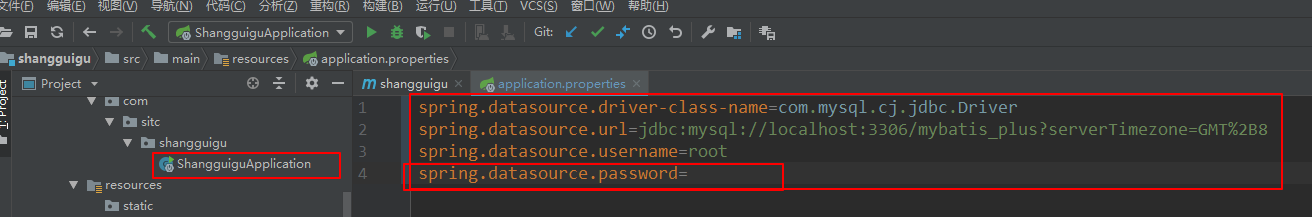


**<!--lombok用来简化实体类，需要安装lombok插件， 这是新的需要学习一下-->**<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
</dependency>

# 配置数据库连接信息：

## 配置完数据库连接信息，

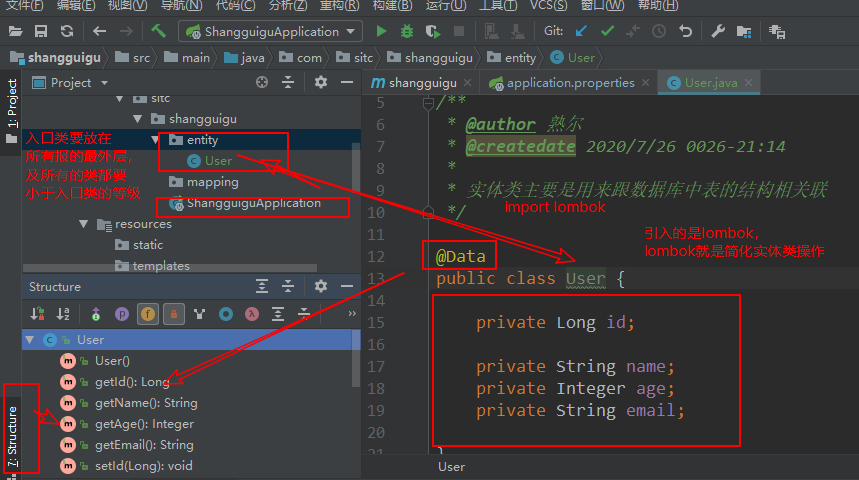
就要开始写数据库中表的对应的实体类结构



其中链接地址中， mybatis\_plus是创建的数据库， 库里面有很多的表， 该库的账号与密码是 root和root

## 实体类层：

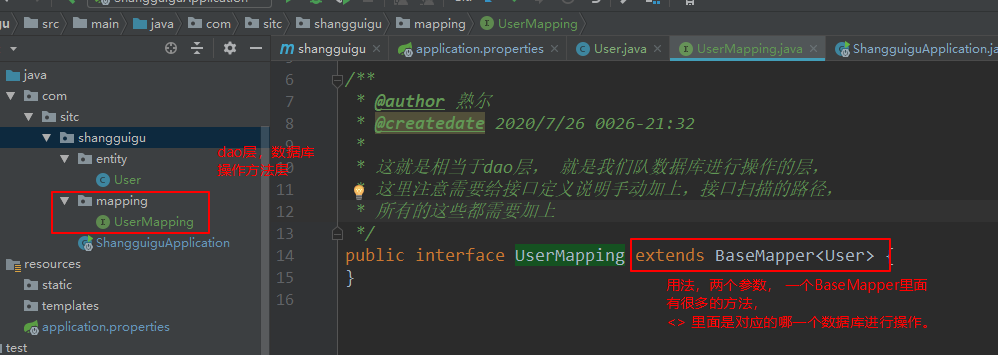
使用lomhok，就省去创建getＸＸＸ和setＸＸＸ的过程。 这个实体类，我们可以这样记忆，就是在数据库中创建表结构的时候， 我们需要在java后台里面有一个对应的映射关系。



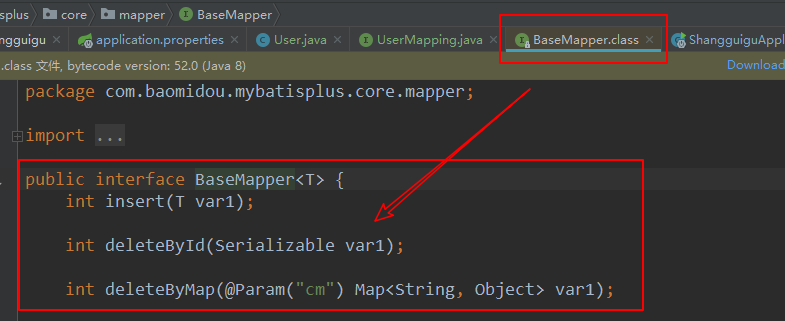
## Dao层

一定要在dao层上添加注解，

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

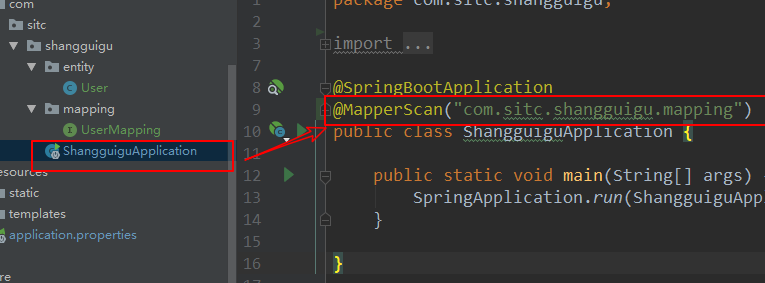


解释一下， 我们看 里面其实就是队数据库操作的各种方法：



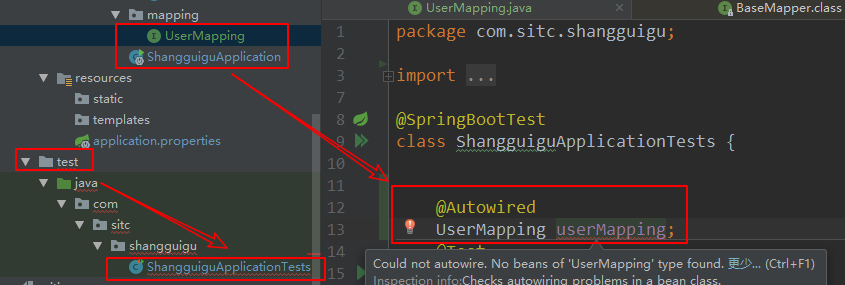
为了能够让入口类，直接扫描到dao层，实际就是我们所说的映射层，需要加上，

@MapperScan(“包名”)



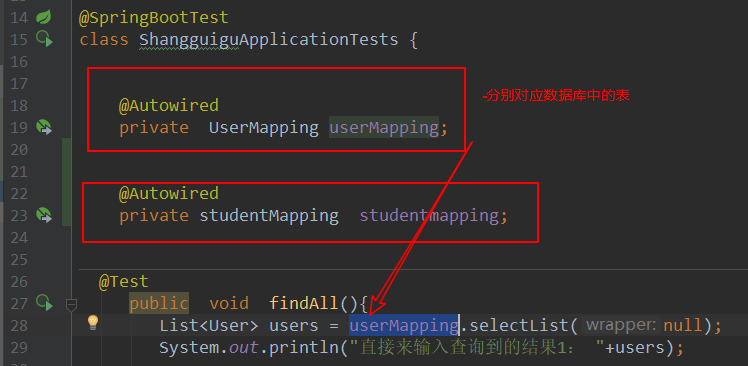
## 测试：

直接使用test来进行测试：



我们对数据操作， 使用的是mapper映射层，dao层里面的方法， @Autowired来进是我们要操作的表，如果需要操作多张表，我们需要多张表，同一哥数据库不同的表都已经引进过来了。

由上图看到，userMapping下面出现红线，是因为，UserMapping是接口， 没有测到bean里面， 解决方法，就是直接添加一个注解后，自动的注册到bean里面。



出现的错误：这是你做了单元测试的时候才会用的到的， 如果你不错测试的，就可以直接在pom文件里面直接上删除掉里面的东西。

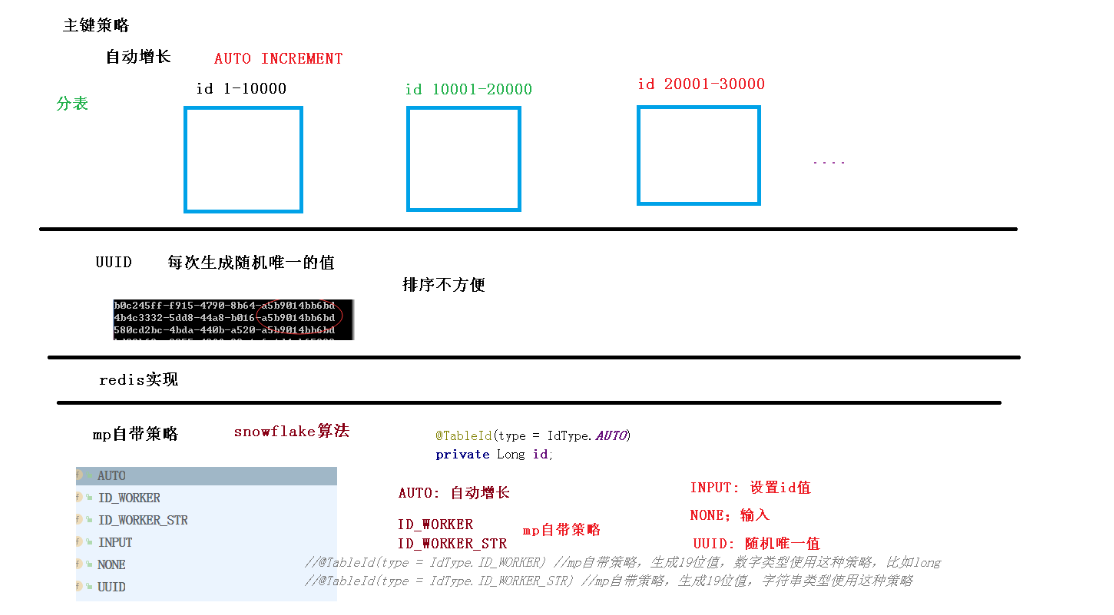


解决方法：在pom文件中添加依赖

<dependency>  
 <groupId>org.junit.platform</groupId>  
 <artifactId>junit-platform-launcher</artifactId>  
 <scope>test</scope>  
</dependency>

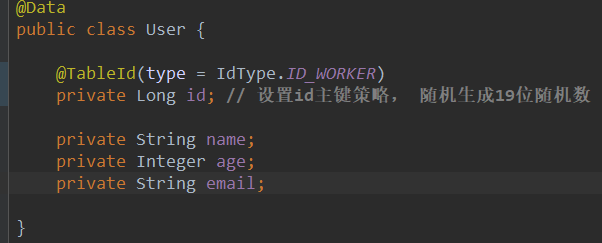
# 主键策略：

这里默认使用的mp，雪花snowflake

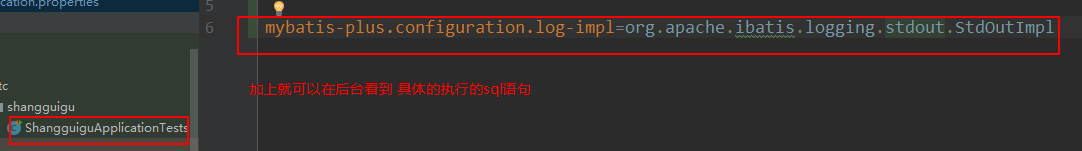


重点是mp， 用法是在private Long id上面加上注解 @TableId（type=IdType.AUTO）

其中ID\_WORK当id的类型为long型的时候才是用，ID\_WORK\_STR表示，id的类型为String的时候，才使用的。



在配置文件里面（设置文件）里面加上 mybatis-plus就可以在后台查看具体的sql语句



# 自动填充数据表里面的时间字段

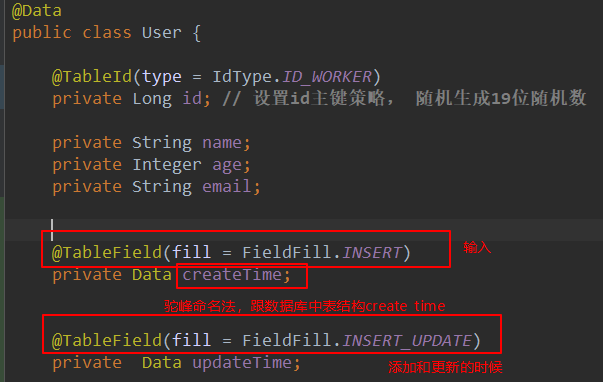
当我们数据库中的字段，有一个是自动添加的比如：创建时间与更新时间的字段，

1. 首先我们需要在数据库中表中加上 create\_time和update\_time两个字段

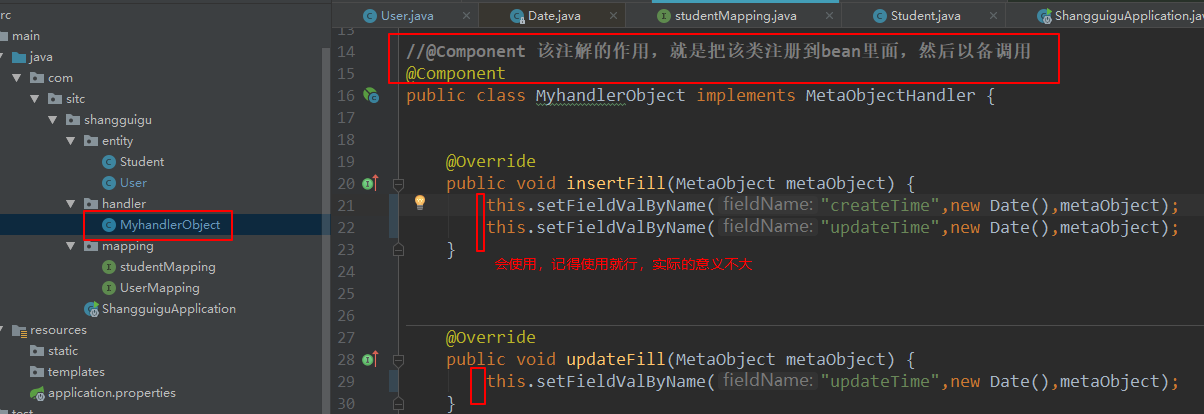


1. 在实体类里面添加相对应的字段名字，然后加上注解，自动填充的属性设置

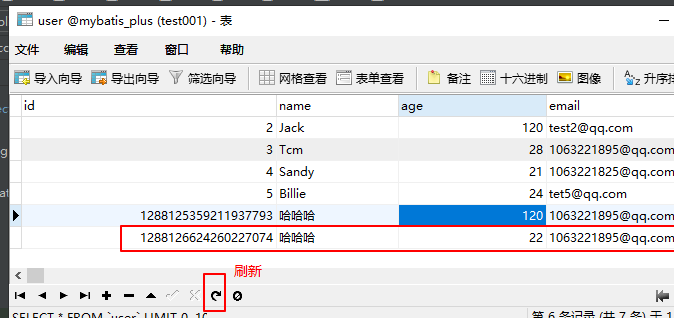
第一步：如果要添加的话，需要在setCreateTime或者setUpdateTime来进行设置值，手动的输入时间，我们使用mp来自动的添加时间；即数据的添加



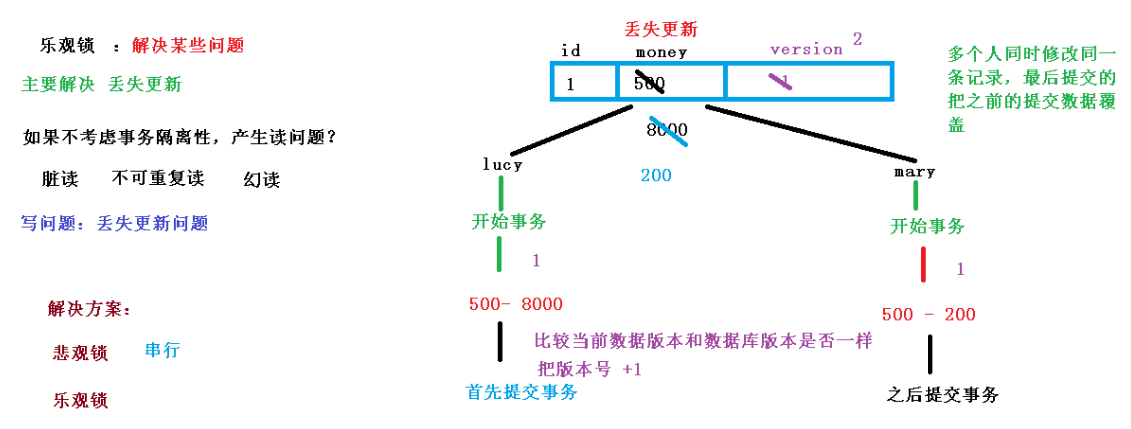
第二部，创建类实现接口metaObjectiveHandler，并实现接口中的方法



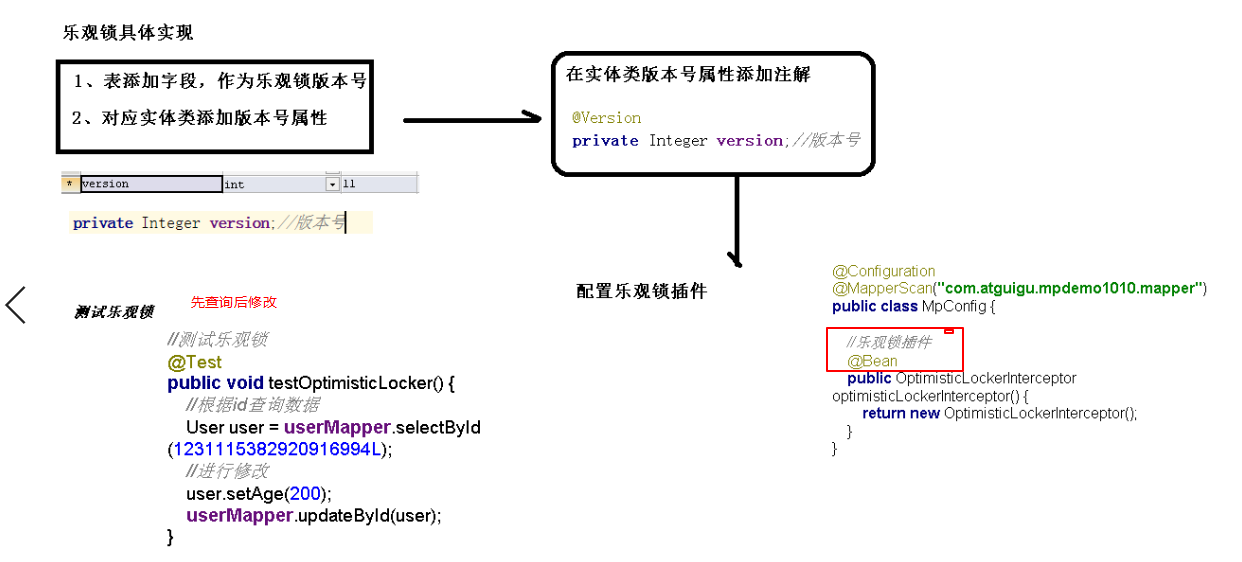




# 乐观锁

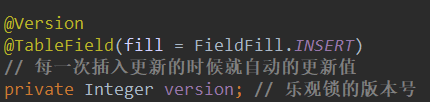


乐观锁就是在表结构中添加一个version字段，用来进行实物提交数据的时候的比较，实物每一次提交事务之前，都需要进行版本的比较，如果不相同，这说明原始的数据已经更改了，需要重新获取，重新操作。悲观锁的意思就是没有version，每一次一个实务操作的时候都需要把表结构锁住。

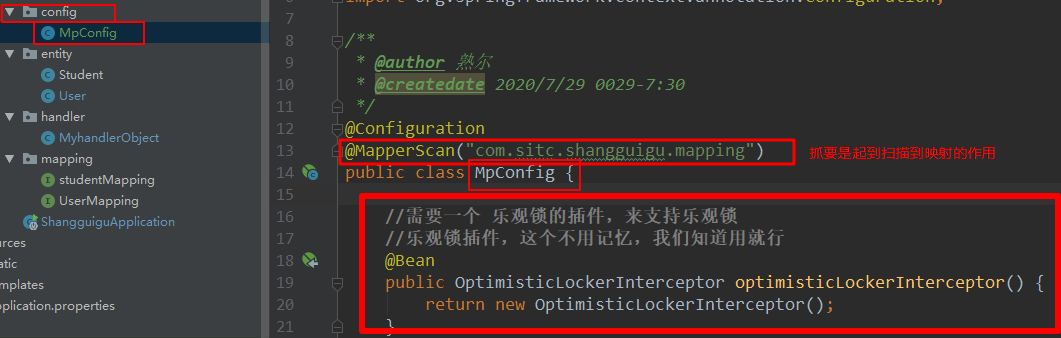


具体的看一下细节：

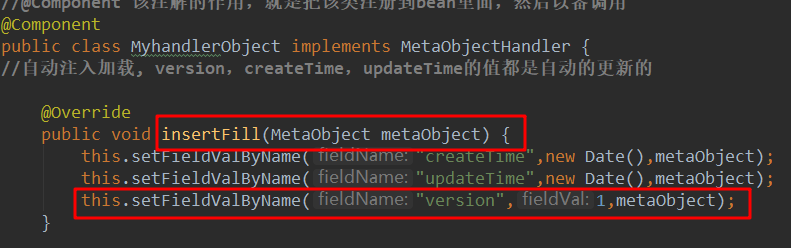
Entity中添加字段，前提是在数据库的表中已经添加了对应的version版本号字段了，我们这里是自动的赋值，每一次插入数据的时候都需要修改

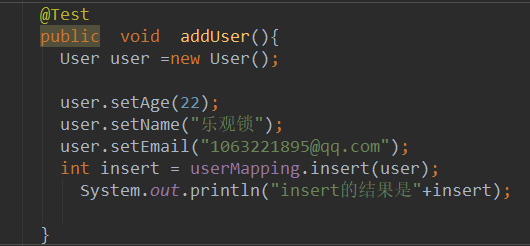


配置文件的配置信息，可以写在启动类里面也可以写在配置文件里面@Configuration的类里面



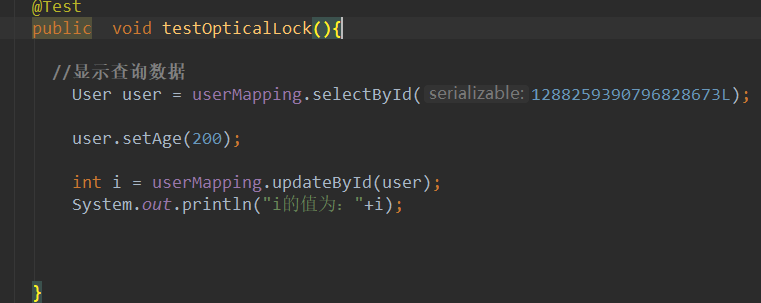
设置起始的数据值加上1





这里的id是设置的直接成成19的数字

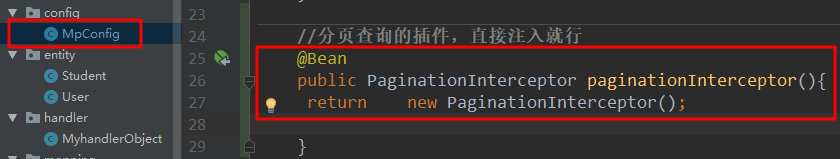




# 分页查询：

首先加入分页查询的插件：



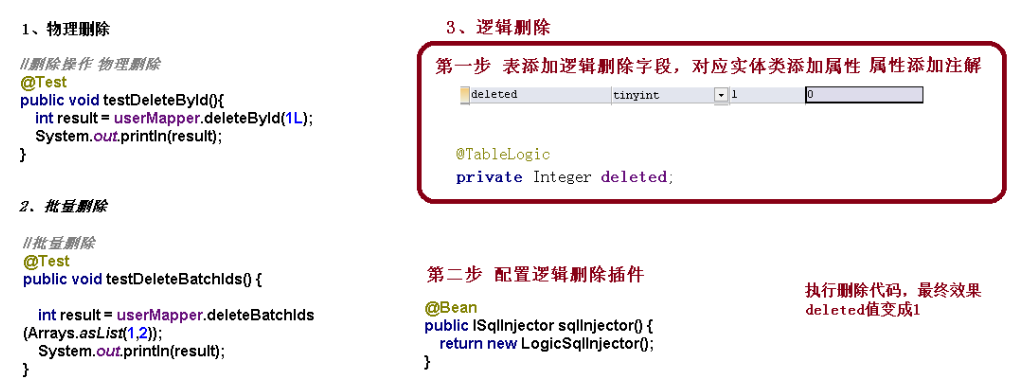


下面是测试的类：：：

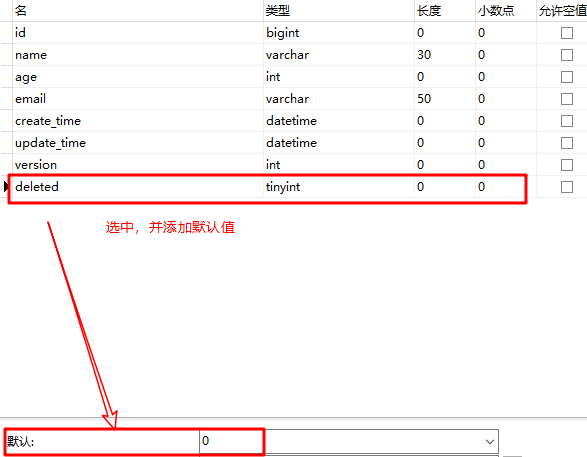
@Test  
public void testPageSearch(){  
  
 **//这个可以在页面上添加一个按钮，然后调用这个方法，就是  
 //页面下面的1页，2页  
   
 //分页查询,注意引入的page是属于哪一个包** Page<User> page = new Page<>(2,3); 这里的2表示第二页，每夜一共三条记录  
  
 **//分页查询到的数据都保存到page里面** userMapping.selectPage(page,null);  
  
 System.*out*.println("当前页信息："+page.getCurrent()); **//**}

# 逻辑删除：

与删除的区别是，不是真正的从数据库里面删掉数据，而是加上一个标识位，那一条数据被删除掉，这条数据对应的标志位的值就变化，这个是可以设定的。



来看具体的操作：



数据库对应的实体类：  
@TableLogic  
private Integer deleted;

配置文件里面添加配置文件的逻辑

**//逻辑删除**@Bean  
public LogicSqlInjector sqlInjector(){  
 return new LogicSqlInjector();  
}

|  |  |  |
| --- | --- | --- |
|  | | 插入的时候，deleted使用默认的值为0 |
|  | |
| @Test public void testLogicSqlDelete(){   int delete = userMapping.deleteById(1288485263332438017L);  System.*out*.println("删除的是："+delete);  } | 逻辑上的删除， 我们从后台可以看到，实际执行的sql语句是update  ==> Preparing: UPDATE user SET deleted=1 WHERE id=? AND deleted=0  ==> Parameters: 1288259390796828673(Long)  由于查询条件为deleted=0，即对于已经删除掉的是不能够逻辑删除的 | |
|  |  | |