# Java逆向工程SpringBoot + Mybatis Generator + MySQL

## Meven pop.xml文件添加引用：

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 <version>8.0.15</version>  
</dependency>

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>2.0.0</version>  
</dependency>

<plugin>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-maven-plugin</artifactId>  
 <version>1.3.7</version>  
 <configuration>  
 <!-- generator 工具配置文件的位置 -->  
 <configurationFile>src/main/resources/mybatis-generator/generatorConfig.xml</configurationFile>  
 <verbose>true</verbose>  
 <overwrite>true</overwrite>  
 </configuration>  
 <dependencies>  
 <dependency>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-core</artifactId>  
 <version>1.3.7</version>  
 </dependency>  
 </dependencies>  
</plugin>

## mybatis-generator配置文件generatorConfig.xml



<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
 "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  
<!-- 配置生成器 -->  
<generatorConfiguration>  
 <!--  
 可以用于加载配置项或者配置文件，在整个配置文件中就可以使用${propertyKey}的方式来引用配置项  
 resource：配置资源加载地址，使用resource，MBG从classpath开始找，比如com/myproject/generatorConfig.properties  
 url：配置资源加载地质，使用URL的方式，比如file:///C:/myfolder/generatorConfig.properties.  
 注意，两个属性只能选址一个;  
 另外，如果用了mybatis-generator-maven-plugin，那在pom.xml中定义的properties都可以直接在generatorConfig.xml中使用  
 -->  
 <!--<properties resource="" url="" />-->  
  
 <!--classPathEntry:数据库的JDBC驱动,换成你自己的驱动位置 可选 -->  
 <classPathEntry location="C:\Users\莫非\.m2\repository\mysql\mysql-connector-java\8.0.15\mysql-connector-java-8.0.15.jar"/>  
  
 <!--  
 context:生成一组对象的环境  
 id:必选，上下文id，用于在生成错误时提示  
 defaultModelType:指定生成对象的样式  
 1，conditional：类似hierarchical；  
 2，flat：所有内容（主键，blob）等全部生成在一个对象中；  
 3，hierarchical：主键生成一个XXKey对象(key class)，Blob等单独生成一个对象，其他简单属性在一个对象中(record class)  
 targetRuntime:  
 1，MyBatis3：默认的值，生成基于MyBatis3.x以上版本的内容，包括XXXBySample；  
 2，MyBatis3Simple：类似MyBatis3，只是不生成XXXBySample；  
 introspectedColumnImpl：类全限定名，用于扩展MBG  
 -->  
 <context id="muphy-tables" targetRuntime="MyBatis3" defaultModelType="flat">  
  
 <!--  
 自动识别数据库关键字，默认false，如果设置为true，根据SqlReservedWords中定义的关键字列表；一般保留默认值，  
 遇到数据库关键字（Java关键字），使用columnOverride覆盖  
 -->  
 <property name="autoDelimitKeywords" value="true"/>  
  
 <!-- 生成的Java文件的编码 -->  
 <property name="javaFileEncoding" value="utf-8"/>  
  
 <!--  
 beginningDelimiter和endingDelimiter：指明数据库的用于标记数据库对象名的符号，  
 比如ORACLE就是双引号，MYSQL默认是`反引号；  
 -->  
 <property name="beginningDelimiter" value="`"/>  
 <property name="endingDelimiter" value="`"/>  
  
 <!-- 格式化java代码 -->  
 <property name="javaFormatter" value="org.mybatis.generator.api.dom.DefaultJavaFormatter"/>  
  
 <!-- 格式化XML代码 -->  
 <property name="xmlFormatter" value="org.mybatis.generator.api.dom.DefaultXmlFormatter"/>  
 <plugin type="org.mybatis.generator.plugins.SerializablePlugin"/>  
 <plugin type="org.mybatis.generator.plugins.ToStringPlugin"/>  
  
 <!-- 注释 -->  
 <commentGenerator>  
 <property name="suppressAllComments" value="false"/><!-- 是否取消注释 -->  
 <property name="suppressDate" value="false"/> <!-- 是否生成注释代时间戳-->  
 </commentGenerator>  
  
 <!-- jdbc连接-->  
 <jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"  
 connectionURL="jdbc:mysql://192.168.18.128/muphy" userId="root"  
 password="...">  
 <!-- 这里面可以设置property属性，每一个property属性都设置到配置的Driver上-->  
 </jdbcConnection>  
  
 <!-- 类型转换 -->  
 <javaTypeResolver>  
 <!-- 是否使用bigDecimal， false可自动转化以下类型（Long, Integer, Short, etc.） -->  
 <property name="forceBigDecimals" value="false"/>  
 </javaTypeResolver>  
  
 <!-- 生成实体类地址 -->  
 <javaModelGenerator targetPackage="me.muphy.recruit.beans" targetProject="src/main/java">  
 <!-- for MyBatis3/MyBatis3Simple  
 自动为每一个生成的类创建一个构造方法，构造方法包含了所有的field；而不是使用setter；  
 -->  
 <property name="constructorBased" value="false"/>  
 <!--  
 在targetPackage的基础上，根据数据库的schema再生成一层package，  
 最终生成的类放在这个package下，默认为false  
 -->  
 <property name="enableSubPackages" value="true"/>  
 <!-- for MyBatis3 / MyBatis3Simple  
 是否创建一个不可变的类，如果为true，  
 那么MBG会创建一个没有setter方法的类，取而代之的是类似constructorBased的类  
 -->  
 <property name="immutable" value="false"/>  
 <!--  
 设置一个根对象，如果设置了这个根对象，那么生成的keyClass或者recordClass会继承这个类；  
 在Table的rootClass属性中可以覆盖该选项  
 注意：如果在key class或者record class中有root class相同的属性，MBG就不会重新生成这些属性了，包括：  
 1，属性名相同，类型相同，有相同的getter/setter方法；  
 -->  
 <!--<property name="rootClass" value="me.muphy.recruit.beans.UserBaseInfo"/>-->  
 <!-- 设置是否在getter方法中，对String类型字段调用trim()方法 -->  
 <property name="trimStrings" value="true"/>  
 </javaModelGenerator>  
  
 <!--  
 生成SQL map的XML文件生成器，  
 注意，在Mybatis3之后，我们可以使用mapper.xml文件+Mapper接口（或者不用mapper接口），  
 或者只使用Mapper接口+Annotation，所以，如果 javaClientGenerator配置中配置了需要生成XML的话，这个元素就必须配置  
 targetPackage/targetProject:同javaModelGenerator  
 -->  
 <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources">  
 <property name="enableSubPackages" value="true"/>  
 </sqlMapGenerator>  
  
 <!--  
 对于mybatis来说，即生成Mapper接口，注意，如果没有配置该元素，那么默认不会生成Mapper接口  
 targetPackage/targetProject:同javaModelGenerator  
 type：选择怎么生成mapper接口（在MyBatis3/MyBatis3Simple下）：  
 1，ANNOTATEDMAPPER：会生成使用Mapper接口+Annotation的方式创建（SQL生成在annotation中），不会生成对应的XML；  
 2，MIXEDMAPPER：使用混合配置，会生成Mapper接口，并适当添加合适的Annotation，但是XML会生成在XML中；  
 3，XMLMAPPER：会生成Mapper接口，接口完全依赖XML；  
 注意，如果context是MyBatis3Simple：只支持ANNOTATEDMAPPER和XMLMAPPER  
 -->  
 <javaClientGenerator targetPackage="me.muphy.recruit.mappers" type="ANNOTATEDMAPPER" targetProject="src/main/java">  
 <!--  
 在targetPackage的基础上，根据数据库的schema再生成一层package，  
 最终生成的类放在这个package下，默认为false  
 -->  
 <property name="enableSubPackages" value="true"/>  
 <!-- 可以为所有生成的接口添加一个父接口，但是MBG只负责生成，不负责检查 -->  
 <!--<property name="rootInterface" value=""/>-->  
 </javaClientGenerator>  
  
 <!--  
 选择一个table来生成相关文件，可以有一个或多个table，必须要有table元素  
 选择的table会生成一下文件：  
 1，SQL map文件  
 2，生成一个主键类；  
 3，除了BLOB和主键的其他字段的类；  
 4，包含BLOB的类；  
 5，一个用户生成动态查询的条件类（selectByExample, deleteByExample），可选；  
 6，Mapper接口（可选）  
 tableName（必要）：要生成对象的表名；  
 注意：大小写敏感问题。正常情况下，MBG会自动的去识别数据库标识符的大小写敏感度，在一般情况下，MBG会  
 根据设置的schema，catalog或tablename去查询数据表，按照下面的流程：  
 1，如果schema，catalog或tablename中有空格，那么设置的是什么格式，就精确的使用指定的大小写格式去查询；  
 2，否则，如果数据库的标识符使用大写的，那么MBG自动把表名变成大写再查找；  
 3，否则，如果数据库的标识符使用小写的，那么MBG自动把表名变成小写再查找；  
 4，否则，使用指定的大小写格式查询；  
 另外的，如果在创建表的时候，使用的""把数据库对象规定大小写，就算数据库标识符是使用的大写，在这种情况下也会使用给定的大小写来创建表名；  
 这个时候，请设置delimitIdentifiers="true"即可保留大小写格式；  
 可选：  
 1，schema：数据库的schema；  
 2，catalog：数据库的catalog；  
 3，alias：为数据表设置的别名，如果设置了alias，那么生成的所有的SELECT SQL语句中，列名会变成：alias\_actualColumnName  
 4，domainObjectName：生成的domain类的名字，如果不设置，直接使用表名作为domain类的名字；  
 可以设置为somepck.domainName，那么会自动把domainName类再放到somepck包里面；  
 5，enableInsert（默认true）：指定是否生成insert语句；  
 6，enableSelectByPrimaryKey（默认true）：指定是否生成按照主键查询对象的语句（就是getById或get）；  
 7，enableSelectByExample（默认true）：MyBatis3Simple为false，指定是否生成动态查询语句；  
 8，enableUpdateByPrimaryKey（默认true）：指定是否生成按照主键修改对象的语句（即update)；  
 9，enableDeleteByPrimaryKey（默认true）：指定是否生成按照主键删除对象的语句（即delete）；  
 10，enableDeleteByExample（默认true）：MyBatis3Simple为false，指定是否生成动态删除语句；  
 11，enableCountByExample（默认true）：MyBatis3Simple为false，指定是否生成动态查询总条数语句（用于分页的总条数查询）；  
 12，enableUpdateByExample（默认true）：MyBatis3Simple为false，指定是否生成动态修改语句（只修改对象中不为空的属性）；  
 13，modelType：参考context元素的defaultModelType，相当于覆盖；  
 14，delimitIdentifiers：参考tableName的解释，注意，默认的delimitIdentifiers是双引号，  
 如果类似MYSQL这样的数据库，使用的是`（反引号，那么还需要设置context的beginningDelimiter和endingDelimiter属性）  
 15，delimitAllColumns：设置是否所有生成的SQL中的列名都使用标识符引起来。默认为false，delimitIdentifiers参考context的属性  
 注意，table里面很多参数都是对javaModelGenerator，context等元素的默认属性的一个复写；  
 -->  
 <table tableName="user\_base\_info" enableCountByExample="false" enableUpdateByExample="false"  
 enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"  
 enableSelectByPrimaryKey="true" enableUpdateByPrimaryKey="true"  
 enableDeleteByPrimaryKey="true">  
 <!-- 参考 javaModelGenerator 的 constructorBased属性-->  
 <property name="constructorBased" value="false"/>  
 <!-- 默认为false，如果设置为true，在生成的SQL中，table名字不会加上catalog或schema； -->  
 <property name="ignoreQualifiersAtRuntime" value="false"/>  
 <!-- 参考 javaModelGenerator 的 immutable 属性 -->  
 <property name="immutable" value="false"/>  
 <!--  
 指定是否只生成domain类，如果设置为true，只生成domain类，  
 如果还配置了sqlMapGenerator，那么在mapper XML文件中，只生成resultMap元素  
 -->  
 <property name="modelOnly" value="false"/>  
 <!-- 参考 javaModelGenerator 的 rootClass 属性-->  
 <!--<property name="rootClass" value=""/>-->  
 <!-- 参考javaClientGenerator 的 rootInterface 属性-->  
 <!--<property name="rootInterface" value=""/>-->  
 <!-- 如果设置了runtimeCatalog，那么在生成的SQL中，使用该指定的catalog，而不是table元素上的catalog-->  
 <!--<property name="runtimeCatalog" value=""/>-->  
 <!-- 如果设置了runtimeSchema，那么在生成的SQL中，使用该指定的schema，而不是table元素上的schema-->  
 <!--<property name="runtimeSchema" value=""/>-->  
 <!-- 如果设置了runtimeTableName，那么在生成的SQL中，使用该指定的tablename，而不是table元素上的tablename-->  
 <!--<property name="runtimeTableName" value=""/>-->  
 <!--  
 注意，该属性只针对MyBatis3Simple有用；  
 如果选择的runtime是MyBatis3Simple，那么会生成一个SelectAll方法  
 如果指定了selectAllOrderByClause，那么会在该SQL中添加指定的这个order条件；  
 -->  
 <!--<property name="selectAllOrderByClause" value="age desc,username asc"/>-->  
 <!--  
 如果设置为true，生成的model类会直接使用column本身的名字，而不会再使用驼峰命名方法  
 比如BORN\_DATE，生成的属性名字就是BORN\_DATE,而不会是bornDate  
 -->  
 <property name="useActualColumnNames" value="false"/>  
 <!--  
 generatedKey用于生成生成主键的方法，  
 如果设置了该元素，MBG会在生成的<insert>元素中生成一条正确的<selectKey>元素，该元素可选  
 column:主键的列名；  
 sqlStatement：要生成的selectKey语句，有以下可选项：  
 Cloudscape:相当于selectKey的SQL为： VALUES IDENTITY\_VAL\_LOCAL()  
 DB2 :相当于selectKey的SQL为： VALUES IDENTITY\_VAL\_LOCAL()  
 DB2\_MF :相当于selectKey的SQL为：SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1  
 Derby :相当于selectKey的SQL为：VALUES IDENTITY\_VAL\_LOCAL()  
 HSQLDB :相当于selectKey的SQL为：CALL IDENTITY()  
 Informix :相当于selectKey的SQL为：select dbinfo('sqlca.sqlerrd1') from systables where tabid=1  
 MySql :相当于selectKey的SQL为：SELECT LAST\_INSERT\_ID()  
 SqlServer :相当于selectKey的SQL为：SELECT SCOPE\_IDENTITY()  
 SYBASE :相当于selectKey的SQL为：SELECT @@IDENTITY  
 JDBC :相当于在生成的insert元素上添加useGeneratedKeys="true"和keyProperty属性  
 -->  
 <generatedKey column="id" sqlStatement="MySql"/>  
 <!--  
 该元素会在根据表中列名计算对象属性名之前先重命名列名，非常适合用于表中的列都有公用的前缀字符串的时候，  
 比如列名为：CUST\_ID,CUST\_NAME,CUST\_EMAIL,CUST\_ADDRESS等；  
 那么就可以设置searchString为"^CUST\_"，并使用空白替换，那么生成的Customer对象中的属性名称就不是  
 custId,custName等，而是先被替换为ID,NAME,EMAIL,然后变成属性：id，name，email；  
 注意，MBG是使用java.util.regex.Matcher.replaceAll来替换searchString和replaceString的，  
 如果使用了columnOverride元素，该属性无效；  
 -->  
 <!--<columnRenamingRule searchString="" replaceString=""/>-->  
 <!--  
 用来修改表中某个列的属性，MBG会使用修改后的列来生成domain的属性；  
 column:要重新设置的列名；  
 注意，一个table元素中可以有多个columnOverride元素哈~  
 -->  
 <columnOverride column="username">  
 <!-- 使用property属性来指定列要生成的属性名称 -->  
 <property name="property" value="userName"/>  
 <!-- javaType用于指定生成的domain的属性类型，使用类型的全限定名-->  
 <!--<property name="javaType" value=""/>-->  
 <!-- jdbcType用于指定该列的JDBC类型 -->  
 <!--<property name="jdbcType" value=""/>-->  
 <!--  
 typeHandler 用于指定该列使用到的TypeHandler，如果要指定，配置类型处理器的全限定名  
 注意，mybatis中，不会生成到mybatis-config.xml中的typeHandler  
 只会生成类似：where id = #{id,jdbcType=BIGINT,typeHandler=com.\_520it.mybatis.MyTypeHandler}的参数描述  
 -->  
 <!--<property name="jdbcType" value=""/>-->  
 <!-- 参考table元素的delimitAllColumns配置，默认为false-->  
 <!--<property name="delimitedColumnName" value=""/>-->  
 </columnOverride>  
 <!--  
 ignoreColumn设置一个MGB忽略的列，如果设置了改列，那么在生成的domain中，生成的SQL中，都不会有该列出现  
 column:指定要忽略的列的名字；  
 delimitedColumnName：参考table元素的delimitAllColumns配置，默认为false  
 注意，一个table元素中可以有多个ignoreColumn元素  
 -->  
 <!--<ignoreColumn column="deptId" delimitedColumnName=""/>-->  
 </table>  
 </context>  
</generatorConfiguration>

## 运行Mybatis Generator插件命令mybatis-generator:generate -e

