#### 添加依赖

<!--Mybatis生成器-->

<dependency>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-core</artifactId>  
 <version>1.3.3</version>  
</dependency>

#### Mybatis generator配置文件generatorConfig.xml

配置数据库连接，Mybatis generator生成model、mapper接口以及mapper.xml的路径。

<?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">

<generatorConfiguration>

<properties resource="generator.properties"/>

<context id="MySqlContext" targetRuntime="MyBatis3" defaultModelType="flat">

<property name="beginningDelimiter" value="`"/>

<property name="endingDelimiter" value="`"/>

<property name="javaFileEncoding" value="UTF-8"/>

<!-- 为模型生成序列化方法-->

<plugin type="org.mybatis.generator.plugins.SerializablePlugin"/>

<!-- 为生成的Java模型创建一个toString方法 -->

<plugin type="org.mybatis.generator.plugins.ToStringPlugin"/>

<!--可以自定义生成model的代码注释-->

<commentGenerator type="com.chance.mall.tiny.mbg.CommentGenerator">

<!-- 是否去除自动生成的注释 true：是 ： false:否 -->

<property name="suppressAllComments" value="true"/>

<property name="suppressDate" value="true"/>

<property name="addRemarkComments" value="true"/>

</commentGenerator>

<!--配置数据库连接-->

<jdbcConnection driverClass="${jdbc.driverClass}"

connectionURL="${jdbc.connectionURL}"

userId="${jdbc.userId}"

password="${jdbc.password}">

<!--解决mysql驱动升级到8.0后不生成指定数据库代码的问题-->

<property name="nullCatalogMeansCurrent" value="true" />

</jdbcConnection>

<!--指定生成model的路径-->

<javaModelGenerator targetPackage="com. chance.mall.tiny.mbg.model" targetProject="src/main/java"/>

<!--指定生成mapper.xml的路径-->

<sqlMapGenerator targetPackage="com. chance.mall.tiny.mbg.mapper" targetProject=" src/main/resources"/>

<!--指定生成mapper接口的的路径-->

<javaClientGenerator type="XMLMAPPER" targetPackage="com. chance.mall.tiny.mbg.mapper"

targetProject="src/main/java"/>

<!--生成全部表tableName设为%-->

<table tableName="pms\_brand">

<generatedKey column="id" sqlStatement="MySql" identity="true"/>

</table>

</context>

</generatorConfiguration>

#### 自定义注释生成model的代码注释

*/\*\**

*\* <p>  
 \* 自定义注释生成器  
 \* <p>  
 \*  
 \** ***@author*** *chance  
 \** ***@since*** *2019-10-23  
 \*/*public class CommentGenerator extends DefaultCommentGenerator {  
  
 private boolean addRemarkComments = false;  
 private static final String *EXAMPLE\_SUFFIX* = "Example";  
 private static final String *API\_MODEL\_PROPERTY\_FULL\_CLASS\_NAME* = "io.swagger.annotations.ApiModelProperty";  
  
 */\*\*  
 \* 设置用户配置的参数  
 \*/* @Override  
 public void addConfigurationProperties(Properties properties) {  
 super.addConfigurationProperties(properties);  
 this.addRemarkComments = StringUtility.*isTrue*(properties.getProperty("addRemarkComments"));  
 }  
  
 */\*\*  
 \* 给字段添加注释  
 \*/* @Override  
 public void addFieldComment(Field field, IntrospectedTable introspectedTable,  
 IntrospectedColumn introspectedColumn) {  
 String remarks = introspectedColumn.getRemarks();  
 //根据参数和备注信息判断是否添加备注信息  
 if (addRemarkComments && StringUtility.*stringHasValue*(remarks)) {  
// addFieldJavaDoc(field, remarks);  
 //数据库中特殊字符需要转义  
 if (remarks.contains("\"")) {  
 remarks = remarks.replace("\"", "'");  
 }  
 //给model的字段添加swagger注解  
 field.addJavaDocLine("@ApiModelProperty(value = \"" + remarks + "\")");  
 }  
 }  
  
 */\*\*  
 \* 给model的字段添加注释  
 \*/* private void addFieldJavaDoc(Field field, String remarks) {  
 //文档注释开始  
 field.addJavaDocLine("/\*\*");  
 //获取数据库字段的备注信息  
 String[] remarkLines = remarks.split(System.*getProperty*("line.separator"));  
 for (String remarkLine : remarkLines) {  
 field.addJavaDocLine(" \* " + remarkLine);  
 }  
 addJavadocTag(field, false);  
 field.addJavaDocLine(" \*/");  
 }  
  
 @Override  
 public void addJavaFileComment(CompilationUnit compilationUnit) {  
 super.addJavaFileComment(compilationUnit);  
 //只在model中添加swagger注解类的导入  
 if (!compilationUnit.isJavaInterface() && !compilationUnit.getType().getFullyQualifiedName().contains(*EXAMPLE\_SUFFIX*)) {  
 compilationUnit.addImportedType(new FullyQualifiedJavaType(*API\_MODEL\_PROPERTY\_FULL\_CLASS\_NAME*));  
 }  
 }  
}

#### 运行Generator类的main函数生成代码

*/\*\**

*\* <p>  
 \* 用于生产MBG的代码  
 \* <p>  
 \*  
 \** ***@author*** *chance  
 \** ***@since*** *2019-10-23  
 \*/*public class Generator {  
  
 public static void main(String[] args) throws Exception {  
 //MBG 执行过程中的警告信息  
 List<String> warnings = new ArrayList<String>();  
 //当生成的代码重复时，覆盖原代码  
 boolean overwrite = true;  
 //读取我们的 MBG 配置文件  
 InputStream is = Generator.class.getResourceAsStream("/generatorConfig.xml");  
 ConfigurationParser cp = new ConfigurationParser(warnings);  
 Configuration config = cp.parseConfiguration(is);  
 is.close();  
  
 DefaultShellCallback callback = new DefaultShellCallback(overwrite);  
 //创建 MBG  
 MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);  
 //执行生成代码  
 myBatisGenerator.generate(null);  
 //输出警告信息  
 for (String warning : warnings) {  
 System.*out*.println(warning);  
 }  
 }  
}