1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE generatorConfiguration
3. PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
4. "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"**>**
5. <!-- 配置生成器 -->
6. **<generatorConfiguration>**
7. <!-- 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
8. resource：配置资源加载地址，使用resource，MBG从classpath开始找，比如com/myproject/generatorConfig.properties
9. url：配置资源加载地质，使用URL的方式，比如file:///C:/myfolder/generatorConfig.properties.
10. 注意，两个属性只能选址一个;
12. 另外，如果使用了mybatis-generator-maven-plugin，那么在pom.xml中定义的properties都可以直接在generatorConfig.xml中使用
13. **<properties** resource="" url="" **/>**
14. --**>**
16. <!-- 在MBG工作的时候，需要额外加载的依赖包
17. location属性指明加载jar/zip包的全路径
18. **<classPathEntry** location="/Program Files/IBM/SQLLIB/java/db2java.zip" **/>**
19. --**>**
21. <!--
22. context:生成一组对象的环境
23. id:必选，上下文id，用于在生成错误时提示
24. defaultModelType:指定生成对象的样式
25. 1，conditional：类似hierarchical；
26. 2，flat：所有内容（主键，blob）等全部生成在一个对象中；
27. 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)
28. targetRuntime:
29. 1，MyBatis3：默认的值，生成基于MyBatis3.x以上版本的内容，包括XXXBySample；
30. 2，MyBatis3Simple：类似MyBatis3，只是不生成XXXBySample；
31. introspectedColumnImpl：类全限定名，用于扩展MBG
32. --**>**
33. **<context** id="mysql" defaultModelType="hierarchical" targetRuntime="MyBatis3Simple" **>**
35. <!-- 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；
36. 一般保留默认值，遇到数据库关键字（Java关键字），使用columnOverride覆盖
37. --**>**
38. **<property** name="autoDelimitKeywords" value="false"**/>**
39. <!-- 生成的Java文件的编码 -->
40. **<property** name="javaFileEncoding" value="UTF-8"**/>**
41. <!-- 格式化java代码 -->
42. **<property** name="javaFormatter" value="org.mybatis.generator.api.dom.DefaultJavaFormatter"**/>**
43. <!-- 格式化XML代码 -->
44. **<property** name="xmlFormatter" value="org.mybatis.generator.api.dom.DefaultXmlFormatter"**/>**
46. <!-- 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； -->
47. **<property** name="beginningDelimiter" value="`"**/>**
48. **<property** name="endingDelimiter" value="`"**/>**
50. <!-- 必须要有的，使用这个配置链接数据库
51. @TODO:是否可以扩展
52. --**>**
53. **<jdbcConnection** driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql:///pss" userId="root" password="admin"**>**
54. <!-- 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 -->
55. **</jdbcConnection>**
57. <!-- java类型处理器
58. 用于处理DB中的类型到Java中的类型，默认使用JavaTypeResolverDefaultImpl；
59. 注意一点，默认会先尝试使用Integer，Long，Short等来对应DECIMAL和 NUMERIC数据类型；
60. --**>**
61. **<javaTypeResolver** type="org.mybatis.generator.internal.types.JavaTypeResolverDefaultImpl"**>**
62. <!--
63. true：使用BigDecimal对应DECIMAL和 NUMERIC数据类型
64. false：默认,
65. scale**>**0;length**>**18：使用BigDecimal;
66. scale=0;length[10,18]：使用Long；
67. scale=0;length[5,9]：使用Integer；
68. scale=0;length**<5**：使用Short；
69. --**>**
70. **<property** name="forceBigDecimals" value="false"**/>**
71. **</javaTypeResolver>**

74. <!-- java模型创建器，是必须要的元素
75. 负责：1，key类（见context的defaultModelType）；2，java类；3，查询类
76. targetPackage：生成的类要放的包，真实的包受enableSubPackages属性控制；
77. targetProject：目标项目，指定一个存在的目录下，生成的内容会放到指定目录中，如果目录不存在，MBG不会自动建目录
78. --**>**
79. **<javaModelGenerator** targetPackage="com.\_520it.mybatis.domain" targetProject="src/main/java"**>**
80. <!--  for MyBatis3/MyBatis3Simple
81. 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；
82. --**>**
83. **<property** name="constructorBased" value="false"**/>**
85. <!-- 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 -->
86. **<property** name="enableSubPackages" value="true"**/>**
88. <!-- for MyBatis3 / MyBatis3Simple
89. 是否创建一个不可变的类，如果为true，
90. 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
91. --**>**
92. **<property** name="immutable" value="false"**/>**
94. <!-- 设置一个根对象，
95. 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
96. 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：
97. 1，属性名相同，类型相同，有相同的getter/setter方法；
98. --**>**
99. **<property** name="rootClass" value="com.\_520it.mybatis.domain.BaseDomain"**/>**
101. <!-- 设置是否在getter方法中，对String类型字段调用trim()方法 -->
102. **<property** name="trimStrings" value="true"**/>**
103. **</javaModelGenerator>**

106. <!-- 生成SQL map的XML文件生成器，
107. 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），
108. 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
109. targetPackage/targetProject:同javaModelGenerator
110. --**>**
111. **<sqlMapGenerator** targetPackage="com.\_520it.mybatis.mapper" targetProject="src/main/resources"**>**
112. <!-- 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 -->
113. **<property** name="enableSubPackages" value="true"**/>**
114. **</sqlMapGenerator>**

117. <!-- 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
118. targetPackage/targetProject:同javaModelGenerator
119. type：选择怎么生成mapper接口（在MyBatis3/MyBatis3Simple下）：
120. 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；
121. 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；
122. 3，XMLMAPPER：会生成Mapper接口，接口完全依赖XML；
123. 注意，如果context是MyBatis3Simple：只支持ANNOTATEDMAPPER和XMLMAPPER
124. --**>**
125. **<javaClientGenerator** targetPackage="com.\_520it.mybatis.mapper" type="ANNOTATEDMAPPER" targetProject="src/main/java"**>**
126. <!-- 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 -->
127. **<property** name="enableSubPackages" value="true"**/>**
129. <!-- 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
130. **<property** name="rootInterface" value=""**/>**
131. --**>**
132. **</javaClientGenerator>**
134. <!-- 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
135. 选择的table会生成一下文件：
136. 1，SQL map文件
137. 2，生成一个主键类；
138. 3，除了BLOB和主键的其他字段的类；
139. 4，包含BLOB的类；
140. 5，一个用户生成动态查询的条件类（selectByExample, deleteByExample），可选；
141. 6，Mapper接口（可选）
143. tableName（必要）：要生成对象的表名；
144. 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
145. 根据设置的schema，catalog或tablename去查询数据表，按照下面的流程：
146. 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；
147. 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；
148. 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；
149. 4，否则，使用指定的大小写格式查询；
150. 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；
151. 这个时候，请设置delimitIdentifiers="true"即可保留大小写格式；
153. 可选：
154. 1，schema：数据库的schema；
155. 2，catalog：数据库的catalog；
156. 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
157. 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；
158. 5，enableInsert（默认true）：指定是否生成insert语句；
159. 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）；
160. 7，enableSelectByExample（默认true）：MyBatis3Simple为false，指定是否生成动态查询语句；
161. 8，enableUpdateByPrimaryKey（默认true）：指定是否生成按照主键修改对象的语句（即update)；
162. 9，enableDeleteByPrimaryKey（默认true）：指定是否生成按照主键删除对象的语句（即delete）；
163. 10，enableDeleteByExample（默认true）：MyBatis3Simple为false，指定是否生成动态删除语句；
164. 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）；
165. 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）；
166. 13，modelType：参考context元素的defaultModelType，相当于覆盖；
167. 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）
168. 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
170. 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；
171. --**>**
172. **<table** tableName="userinfo" **>**
174. <!-- 参考 javaModelGenerator 的 constructorBased属性-->
175. **<property** name="constructorBased" value="false"**/>**
177. <!-- 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； -->
178. **<property** name="ignoreQualifiersAtRuntime" value="false"**/>**
180. <!-- 参考 javaModelGenerator 的 immutable 属性 -->
181. **<property** name="immutable" value="false"**/>**
183. <!-- 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 -->
184. **<property** name="modelOnly" value="false"**/>**
186. <!-- 参考 javaModelGenerator 的 rootClass 属性
187. **<property** name="rootClass" value=""**/>**
188. --**>**
190. <!-- 参考javaClientGenerator 的  rootInterface 属性
191. **<property** name="rootInterface" value=""**/>**
192. --**>**
194. <!-- 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
195. **<property** name="runtimeCatalog" value=""**/>**
196. --**>**
198. <!-- 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
199. **<property** name="runtimeSchema" value=""**/>**
200. --**>**
202. <!-- 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
203. **<property** name="runtimeTableName" value=""**/>**
204. --**>**
206. <!-- 注意，该属性只针对MyBatis3Simple有用；
207. 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；
208. --**>**
209. **<property** name="selectAllOrderByClause" value="age desc,username asc"**/>**
211. <!-- 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 -->
212. **<property** name="useActualColumnNames" value="false"**/>**

215. <!-- generatedKey用于生成生成主键的方法，
216. 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
217. column:主键的列名；
218. sqlStatement：要生成的selectKey语句，有以下可选项：
219. Cloudscape:相当于selectKey的SQL为： VALUES IDENTITY\_VAL\_LOCAL()
220. DB2       :相当于selectKey的SQL为： VALUES IDENTITY\_VAL\_LOCAL()
221. DB2\_MF    :相当于selectKey的SQL为：SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1
222. Derby      :相当于selectKey的SQL为：VALUES IDENTITY\_VAL\_LOCAL()
223. HSQLDB      :相当于selectKey的SQL为：CALL IDENTITY()
224. Informix  :相当于selectKey的SQL为：select dbinfo('sqlca.sqlerrd1') from systables where tabid=1
225. MySql      :相当于selectKey的SQL为：SELECT LAST\_INSERT\_ID()
226. SqlServer :相当于selectKey的SQL为：SELECT SCOPE\_IDENTITY()
227. SYBASE      :相当于selectKey的SQL为：SELECT @@IDENTITY
228. JDBC      :相当于在生成的insert元素上添加useGeneratedKeys="true"和keyProperty属性
229. **<generatedKey** column="" sqlStatement=""**/>**
230. --**>**
232. <!--
233. 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，
234. 比如列名为：CUST\_ID,CUST\_NAME,CUST\_EMAIL,CUST\_ADDRESS等；
235. 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
236. custId,custName等，而是先被替换为ID,NAME,EMAIL,然后变成属性：id，name，email；
238. 注意，MBG是使用java.util.regex.Matcher.replaceAll来替换searchString和replaceString的，
239. 如果使用了columnOverride元素，该属性无效；
241. **<columnRenamingRule** searchString="" replaceString=""**/>**
242. --**>**

245. <!-- 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；
246. column:要重新设置的列名；
247. 注意，一个table元素中可以有多个columnOverride元素哈~
248. --**>**
249. **<columnOverride** column="username"**>**
250. <!-- 使用property属性来指定列要生成的属性名称 -->
251. **<property** name="property" value="userName"**/>**
253. <!-- javaType用于指定生成的domain的属性类型，使用类型的全限定名
254. **<property** name="javaType" value=""**/>**
255. --**>**
257. <!-- jdbcType用于指定该列的JDBC类型
258. **<property** name="jdbcType" value=""**/>**
259. --**>**
261. <!-- 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
262. 注意，mybatis中，不会生成到mybatis-config.xml中的typeHandler
263. 只会生成类似：where id = #{id,jdbcType=BIGINT,typeHandler=com.\_520it.mybatis.MyTypeHandler}的参数描述
264. **<property** name="jdbcType" value=""**/>**
265. --**>**
267. <!-- 参考table元素的delimitAllColumns配置，默认为false
268. **<property** name="delimitedColumnName" value=""**/>**
269. --**>**
270. **</columnOverride>**
272. <!-- 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
273. column:指定要忽略的列的名字；
274. delimitedColumnName：参考table元素的delimitAllColumns配置，默认为false
276. 注意，一个table元素中可以有多个ignoreColumn元素
277. **<ignoreColumn** column="deptId" delimitedColumnName=""**/>**
278. --**>**
279. **</table>**
281. **</context>**
283. **</generatorConfiguration>**