链滴

一文解析 MyBatis Generator 的使用及配置

作者：Deecyn

原文链接：https://ld246.com/article/1586269892542

来源网站：链滴

MyBatis-Generator 是 MyBatis 提供的一个代码生成工具，可以帮助我们生成数据库表对应的持久对象（也称作 Model、PO）、操作数据库的接口（dao）、简单 SQL 的 mapper（XML 形式或注形式）。

MyBatis-Generator （常简写为 MBG 或 mbg）是一个独立工具，你可以下载它的 jar 包来运行，可以在 Ant 和 Maven 中运行。其官方网址为：https://mybatis.org/generator/

一、引入 MyBatis-Generator 及环境

对于这篇博文，我是在基于 SpringBoot 的 **Maven 项目**环境中配置、使用和讲解的，使用的 IDE 是 ntelliJ IDEA。

既然需要使用 MyBatis-Generator，那么在项目中就一定使用了 MyBatis 和某一种数据库，并且这依赖应该已经在 Maven 中配置好了。例如 pom 文件中的配置：

```
<dependencies>
  <!-- 为了方便，不展示其它配置... -->
  <dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
  </dependency>
</dependencies>
```

对于在 Maven 项目中引入 MyBatis-Generator，这里介绍**两种方式**，具体需要用哪种方式取决于你用哪种方式来运行 MyBatis-Generator 工具： （对于如何运行 MyBatis-Generator 工具，参见本文三节）

1. 方式一：在 Maven 中导入依赖

这种方式适用于：用 Java 代码来运行 MyBatis-Generator 工具。在 pom 文件中引入 mybatis-generator-core 依赖：

```
<dependencies>
  <!-- 为了方便，不展示其它配置... -->
  <dependency>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-core</artifactId>
    <version>1.3.7</version>
  </dependency>
</dependencies>
```

2. 方式二：在 Maven 中引入插件

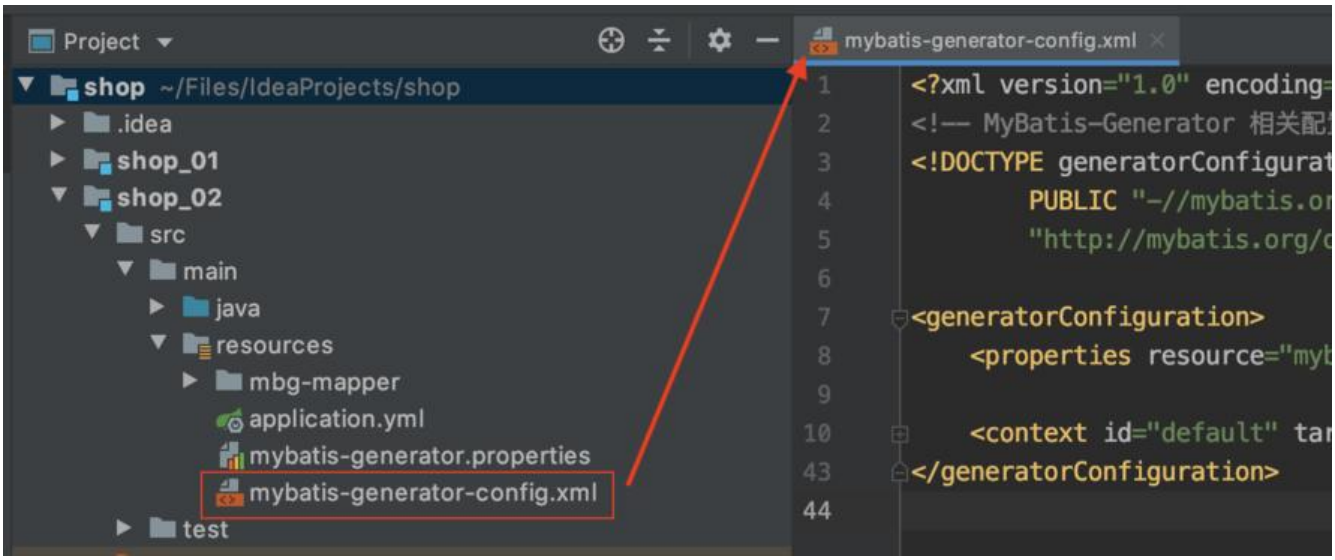
这种方式适用于：通过 Maven 项目的 Plugins 或 Maven 的命令行来运行 MyBatis-Generator 工具在 pom 文件中引入 mybatis-generator-maven-plugin 插件：

```
<build>
```

```
    <!-- 为了方便，不展示其它配置... -->
    <plugins>
      <plugin>
      <groupId>org.mybatis.generator</groupId>
      <artifactId>mybatis-generator-maven-plugin</artifactId>
      <version>1.3.7</version>
    </plugin>
  </plugins>
</build>
```

二、MyBatis-Generator 配置文件

MyBatis-Generator 需要一个 xml 配置文件，来详细配置生成代码的各种细节。例如，在**项目的 resources 目录下**新建一个 mybatis-generator-config.xml 配置文件：



其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MyBatis-Generator 相关配置 -->
<!DOCTYPE generatorConfiguration
       PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
       "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<!-- 所有的配置均在根元素 generatorConfiguration 下 -->
<generatorConfiguration>
  ...
  ...
</generatorConfiguration>
```

在该文件中，所有的配置均在根元素 generatorConfiguration 下。根元素 generatorConfiguration 有 3 个子元素可供配置，这 3 个子元素必须按照下面给出的**次数和顺序**进行配置：（**没错，MyBatis Generator 对配置的顺序也有严格的要求**）

1. **properties** (0 or 1)，可出现 0 次或 1 次。
2. **classPathEntry** (0...N)，可出现 0 次或多次。
3. **context** (1...N)，至少出现 1 次。

其中，properties 和 classPathEntry 元素用于引入外部配置或文件；context 是核心元素，里面包有各种详细配置。

1. 引入外部配置文件

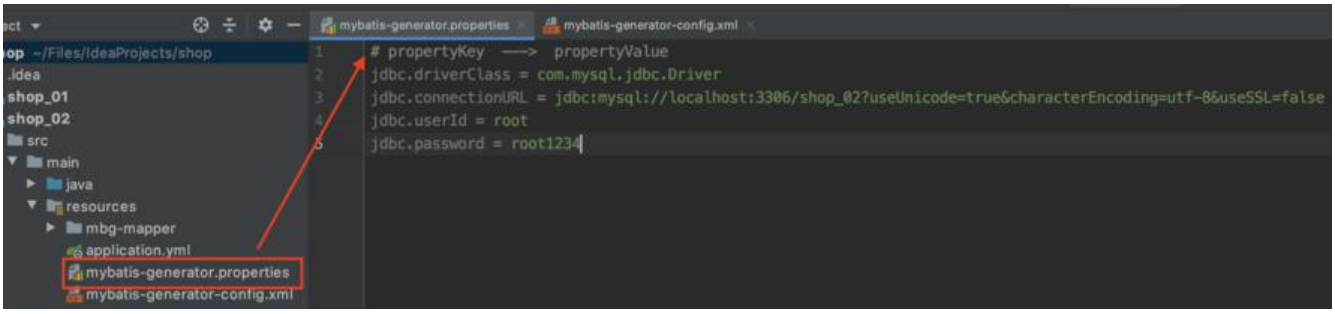
在配置 MyBatis-Generator 时，是可以引入外部配置或文件的。

properties 元素

元素 properties 可以用于加载外部配置项或配置文件，该元素有两个属性，均用来指定外部配置文的地址：

- resource：使 MBG 从 classpath 开始查找；一般可以使用相对于当前 xml 配置文件的相对路径。
- url：采用 URL 的方式；例如可以使用 file 协议 file:///Users/deecyn/Files/mybatis-generator.roperties 从计算机本地查找，也可以使用 http 协议在互联网上查找，等等。

注意，这两个属性**只能选择一个**使用。例如，引入下图中的 mybatis-generator.properties 配置文：



用于引入文件的代码如下：

```
<!-- 引入外部配置文件 -->
<properties resource="mybatis-generator.properties"/>
```

之后在整个 xml 配置文件中就可以通过 ${propertyKey} 的方式来引用配置项。

classPathEntry 元素

使用 classPathEntry 元素，可以在 MBG 工作的时候，加载额外需要的依赖包。其中，location 属指明需要加载的 jar/zip 包的全路径。例如：

```
<!-- 加载需要的额外的依赖包 -->
<classPathEntry location="/Users/deecyn/Files/db2java.zip"/>
```

2. 配置 context 核心元素

在 generationConfiguration 的子元素中，context 是核心元素，用于配置生成一组对象的环境。元 context 有 4 个属性可供配置：

- **id**，必填，上下文 id，用于在生成错误时提示；保证多个 context 的 id 不重复就行。
- **targetRuntime**，选填项，这个配置会影响生成的 dao 和 mapper.xml 的内容。常见值为：

1. MyBatis3，默认值，生成基于 MyBatis 3.x 以上版本的内容，包括很多类似 XxxByExample 的 dao 方法。

2. MyBatis3Simple，类似 MyBatis3，只是不生成类似 XxxByExample 的 dao 方法， **一般选不生成这些繁杂的方法。**

3. 还有其它可配置的值，详情见 官网 。

- **defaultModelType**，选填项，用于指定生成对象的样式。其值为：

1. conditional，默认值，类似于 hierarchical。区别是，不会为只有一个字段的数据库表生成一单独的类。

2. hierarchical，主键生成一个 XxxKey 对象（key class），Blob 等字段单独生成一个对象，其简单属性在一个对象中（record class）。

3. flat，所有字段（主键、blob 等）全部生成在一个对象中。

- **introspectedColumnImpl**，选填项，类全限定名，用于 扩展 MBG 。

示例配置如下：

```
<context id="MySqlContext" targetRuntime="MyBatis3" defaultModelType="flat">
  ...
</context>
```

context 的子元素

元素 context 中，有多个子元素需要配置。同样的，context 的子元素必须按照下面给出的**次数和顺序**行配置：

1. **property** (0...N)
2. **plugin** (0...N)
3. **commentGenerator** (0 or 1)
4. **connectionFactory** 和 **jdbcConnection**，二选一进行配置
5. **javaTypeResolver** (0 or 1)
6. **javaModelGenerator**（有且仅有 1 次）
7. **sqlMapGenerator** (0 or 1)
8. **javaClientGenerator** (0 or 1)
9. **table** (1...N)

可以看出，javaModelGenerator、table 以及 connection 元素的配置是必需的。

property 元素

用于为代码生成指定属性，或为其它元素指定属性。可以配置零个或多个，常见的 property 配置如：

```
<!-- 自动识别数据库关键字，默认为 false，一般保留默认值，遇到数据库关键字（Java关键字）时按照 table 元素中 columnOverride 属性的配置进行覆盖；
  如果设置为 true， 则需按照 SqlReservedWords 中定义的关键字列表，对关键字进行定界（分隔;
```

```
 定界符（分隔符）参见 beginningDelimiter 和 endingDelimiter 的设置-->
<property name="autoDelimitKeywords" value="false"/>

<!-- beginningDelimiter 和 endingDelimiter，定界符（分隔符），指明用于标记数据库关键字的
号，默认为为双引号 (");
 在 oracle 中是双引号 (")，在 MySQL 中需配置为反引号 (`) -->
<property name="beginningDelimiter" value="`"/>
<property name="endingDelimiter" value="`"/>

<!-- 生成的 Java 文件的编码  -->
<property name="JavaFileEncoding" value="UTF-8"/>

<!-- 格式化 Java 代码 -->
<property name="javaFormatter" value="org.mybatis.generator.api.dom.DefaultJavaFormatte
"/>
<!-- 格式化 XML 代码 -->
<property name="xmlFormatter" value="org.mybatis.generator.api.dom.DefaultXmlFormatte
"/>
```

注：SqlReservedWords 关键字列表

plugin 元素

配置插件，可以有零个或多个，常见的 plugin 配置有：

```
<!-- 使生成的 Model 实现 Serializable 接口  -->
<plugin type="org.mybatis.generator.plugins.SerializablePlugin"/>
<!--  为生成的 Model 覆写 toString() 方法 -->
<plugin type="org.mybatis.generator.plugins.ToStringPlugin"/>
<!--  为生成的 Model 覆写 equals() 和 hashCode() 方法 -->
<plugin type="org.mybatis.generator.plugins.EqualsHashCodePlugin"/>
```

commentGenerator 元素

可以配置 0 个或 1 个，用来配置生成的注释，默认是生成注释的，并且会在注释中添加时间等信息。果想沿用默认的注释配置的话，可以不用配置 commentGenerator 元素。否则，可以进行如下配置：

```
<commentGenerator>
  <!--  不生成所有注释，默认为 false  -->
  <property name="suppressAllComments" value="true"/>

  <!--  生成的注释中不包含时间信息，默认为 false -->
  <property name="suppressDate" value="true"/>
  <!--  生成的注释中，时间的显示格式 -->
  <property name="dateFormat" value="yyyy/MM/dd"/>
  <!-- 是否添加数据库表中字段的注释，默认为 false  -->
  <property name="addRemarkComments" value="true"/>
</commentGenerator>
```

当然，你也可以 自定注释生成器 。

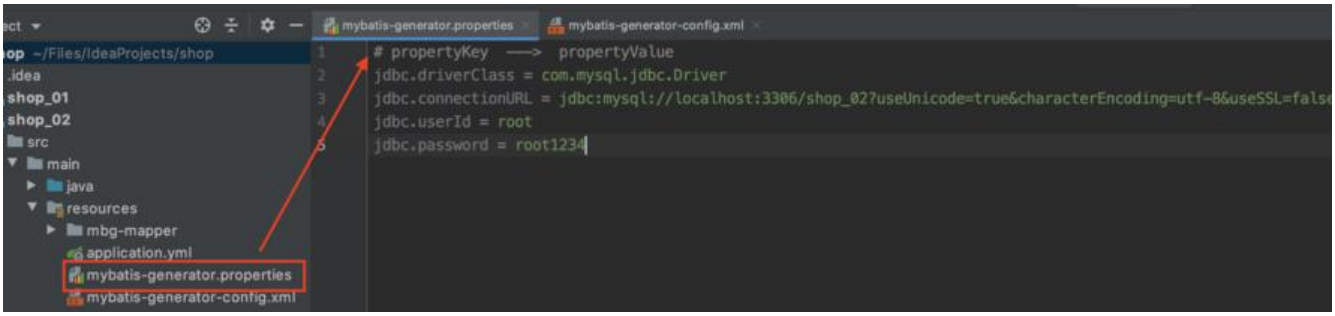
jdbcConnection 元素

配置数据库连接，具体如下：

```
<!-- 配置数据库连接 -->
<jdbcConnection driverClass="${jdbc.driverClass}"
                connectionURL="${jdbc.connectionURL}"
                userId="${jdbc.userId}"
                password="${jdbc.password}">

<!-- 若为 8.0 版本以上的 mysql-connector-java 驱动，需要设置 nullCatalogMeansCurrent = true
-->
  <!-- <property name="nullCatalogMeansCurrent" value="true"/> -->
</jdbcConnection>
```

其中，${propertyKey} 里面是引用的外部配置文件中的 propertyValue：



你也可以写死，那么就不用在 <properties resource=""/> 中引入此文件了。

这里面值得注意的是 <property name="nullCatalogMeansCurrent" value="true"/> ，当 mysql-onnector-java 驱动在 8.0 版本以上时，如果不配置这一项为 true，会不生成指定数据库中表的 Mapper。具体原因可参考文章：MyBatis Generator踩坑与自救 。

javaTypeResolver 元素

可以配置 0 或 1 个，用来配置 JDBC 到 Java 中的类型转换规则，如果不进行配置，则使用默认的转规则，默认使用 org.mybatis.generator.internal.types.JavaTypeResolverDefaultImpl。

就算要配置，也只能配置 BigDecimal 和时间类型的转换：

```
<javaTypeResolver>
  <!-- 是否强制使用 BigDecimal;
         默认为 false，把 JDBC 的 DECIMAL 和 NUMERIC 类型解析为 Integer;
         设置为 true 时，把 JDBC 的 DECIMAL 和 NUMERIC 类型解析为 java.math.BigDecimal
       -->
  <property name="forceBigDecimals" value="true"/>

  <!-- 设置时间类型的转换,
         默认 false，将所有 JDBC 的时间类型解析为 java.util.Date;
    设置为 true 时，将 JDBC 的时间类型按如下规则解析:
      DATE        -> java.time.LocalDate
      TIME        -> java.time.LocalTime
      TIMESTAMP       -> java.time.LocalDateTime
      TIME_WITH_TIMEZONE       -> java.time.OffsetTime
      TIMESTAMP_WITH_TIMEZONE      -> java.time.OffsetDateTime
    -->
  <property name="useJSR310Types" value="true"/>
```

```
</javaTypeResolver>
```

javaModelGenerator 元素

Java 模型生成器，有且仅能配置一个，负责 key 类（见 context 元素的 defaultModelType 属性）Java Bean 实体类、查询类的生成。

元素 javaModelGenerator 有两个属性：

- targetPackage：生成的类要放的包，具体的包受子元素 enableSubPackages 影响；
- targetProject：目标项目，指定一个已存在的目录。（targetProject 的路径配置，对于不同的 MB 启动方式会有一些区别，详情见第三节：运行 MyBatis-Generator）

在 javaModelGenerator 元素中还可以配置多个 property 子元素，具体代码如下：

```
<!--  配置 Java 模型生成器 -->
<javaModelGenerator targetPackage="deecyn.shop_02.mbg.model" targetProject="src/main/
ava">
  <!-- 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；
    默认值为 false -->
  <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="false"/>

  <!-- 是否创建一个不可变的类：如果为true，那么 MBG 生成的类会没有 setter 方法，
    采用构造函数的方式来接收和设置每个字段的值，此时会忽略 constructorBased 属性的设置；
    默认值为 false  -->
  <property name="immutable" value="false"/>

  <!-- 设置在 getter 方法中，是否对 String 类型的字段调用 trim() 方法；默认为 false -->
  <property name="trimStrings" value="true"/>
</javaModelGenerator>
```

sqlMapGenerator 元素

可以配置 0 或 1 个，生成 SQL Map 的 xml 文件生成器。在 MyBatis3 之后，我们可以使用 mapper.ml 文件 + Mapper 接口，或者只使用 Mapper 接口 + Annotation；所以，如果 javaClientGenerator 元素中配置了需要生成 xml 的话，这个元素就必须配置。

该元素有 targetPackage 和 targetProject 两个属性，原理与 javaModelGenerator 元素的相同，不过这里指的是 resource 目录下存放 mapper.xml 文件的路径。具体代码如下：

```
<!-- SQL Map 的 xml 文件生成器 -->
<sqlMapGenerator targetPackage="mbg-mapper" targetProject="src/main/resources">
  <!-- 同 javaModelGenerator 元素中的配置  -->
  <property name="enableSubPackages" value="false"/>
</sqlMapGenerator>
```

javaClientGenerator 元素

可以配置 0 或 1 个，用于配置关于 Mapper 接口的生成，如果没有配置该元素，那么默认不会生成 apper 接口。

元素 javaClientGenerator 有 3 个属性，其中 targetPackage 和 targetProject 属性的配置与 javaM delGenerator 元素的原理相同，只不过这里指的是 java 目录下存放 Mapper 接口的路径。关于 type 属性，有 3 个可选值：

- **ANNOTATEDMAPPER**，按照使用 Mapper 接口 + Annotation 的方式生成文件，SQL 生成在 应的 Annotation 中，不会生成 xml 文件。
- **MIXEDMAPPER**，使用混合配置，会生成 Mapper 接口，并适当添加合适的 Annotation，也会有 SQL 生成在 XML 文件中。
- **XMLMAPPER**，会生成 Mapper 接口，接口完全依赖 XML 文件。

注意，如果 context 元素的 defaultModelType 属性设置为 MyBatis3Simple，那么就只支持 ANN TATEDMAPPER 和 XMLMAPPER 的方式。**一般建议将 type 设置成 XMLMAPPER。**

```
<!--  关于 Mapper 接口的生成 -->
<javaClientGenerator type="XMLMAPPER" targetPackage="deecyn.shop_02.mbg.mapper"
                     targetProject="src/main/java">
  <!-- 同 javaModelGenerator 元素中的配置  -->
  <property name="enableSubPackages" value="false"/>
</javaClientGenerator>
```

table 元素

一个 table 元素对应一张数据库表，如果想同时为多张表生成代码，需要配置多个 table 元素；或者以将 tableName 设置为 % 来为全部表生成代码。

元素 table 除开一个必须的属性 **tableName**（数据库表名称）需要设置外，还有很多**可选**的属性，分属性如下：

1. schema，数据库的 schema；
2. catalog，数据库的 catalog；
3. domainObjectName：生成的 domain 类的名字，如果不设置，直接使用表名的驼峰命名作为 d main 类的名字；可以设置为 somepackage.domainName，那么会自动把 domainName 类再放到 omepackage 包里面；
4. enableSelectByExample，默认 true，MyBatis3Simple 为 false，指定是否生成动态查询语句；
5. enableUpdateByPrimaryKey，默认 true，指定是否生成按照主键修改对象的语句（即 update）；
6. enableDeleteByExample，默认 true，MyBatis3Simple 为 false，指定是否生成动态删除语句；
7. enableCountByExample，默认 true，MyBatis3Simple 为 false，指定是否生成动态查询总条数 句（用于分页的总条数查询）；
8. enableUpdateByExample，默认 true，MyBatis3Simple为false，指定是否生成动态修改语句（ 修改对象中不为空的属性）；
9. modelType，参考 context 元素的 defaultModelType，相当于对其进行覆盖。

此外，table 元素中还可以配置多个 property 和 columnOverride 等子元素。示例代码如下：

```
<!-- 配置需要生成代码的数据库表 -->
<table tableName="pms_brand" domainObjectName="PmsBrand"
```

```
      enableCountByExample="false" enableUpdateByExample="false"
      enableDeleteByExample="false" enableSelectByExample="false"
      selectByExampleQueryId="false">

  <!-- 指定是否只生成 domain 类，默认为 false；
    如果设置为 true，则只生成 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"/>

  <!-- 默认为 false；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
用驼峰命名方法。比如 CREATE_DATE，生成的属性名字就是 CREATE_DATE,而不会是 createDate -
>
  <property name="useActualColumnNames" value="false"/>

  <!-- 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 <sel
ctKey> 元素 -->
  <generatedKey column="id" sqlStatement="MySql" identity="true"/>

  <!-- 用来修改表中某个列的属性，MBG 会根据修改后的配置来生成 domain 的属性；
    column：要重新设置的列名；一个 table 元素中可以定义多个 columnOverride 元素哈 -->
  <columnOverride column="show_status">
    <!-- 使用 property 属性来指定列要生成的属性名称 -->
    <property name="property" value="showStatus"/>

    <!-- 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="java.lang.Integer"/>

    <!-- jdbcType用于指定该列的JDBC类型
    <property name="jdbcType" value=""/>
    -->
  </columnOverride>
</table>
```

三、运行 MyBatis-Generator

对于 MyBatis-Generator，不同的运行方式，对项目和文件的配置会有一些区别，本文介绍两种运行 MBG 的方式。

方式一：使用 Java 代码编程运行

通过这种方式运行 MBG，在本文的第一节，引入 MyBatis-Generator 时，需要按照方式一在 Maven 的 pom 文件中引入依赖：

```
<dependencies>
  <!-- 为了方便，不展示其它配置... -->
  <dependency>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-core</artifactId>
    <version>1.3.7</version>
  </dependency>
</dependencies>
```

然后在项目中新建一个 Java 类，代码类似下面：

```
package deecyn.shop_02.mbg;

import org.mybatis.generator.api.MyBatisGenerator;
import org.mybatis.generator.config.Configuration;
import org.mybatis.generator.config.xml.ConfigurationParser;
import org.mybatis.generator.internal.DefaultShellCallback;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

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("/mybatis-generator-config.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);
        }
    }
}
```

运行类中的 main() 方法即可在相应的目录下生成对应的代码。

需要注意的是，当你的项目中有多个 Module 时，在配置 javaModelGenerator、sqlMapGenerator 和 javaClientGenerator 元素的 targetProject 属性时，需要在前面加上当前的 Module 名称。例如前的 Module 名称为 shop_02 时：

```
<javaModelGenerator targetPackage="deecyn.shop_02.mbg.model" targetProject="shop_02/
rc/main/java"/>
<sqlMapGenerator targetPackage="mbg-mapper" targetProject="shop_02/src/main/resourc
s"/>
<javaClientGenerator type="XMLMAPPER" targetPackage="deecyn.shop_02.mbg.mapper"
                     targetProject="shop_02/src/main/java"/>
```

否则会提示找不到对应的 java 和 resource 目录。

方式二：通过 Maven 插件运行

通过这种方式运行 MBG，在本文的第一节，引入 MyBatis-Generator 时，需要按照方式二在 Maven

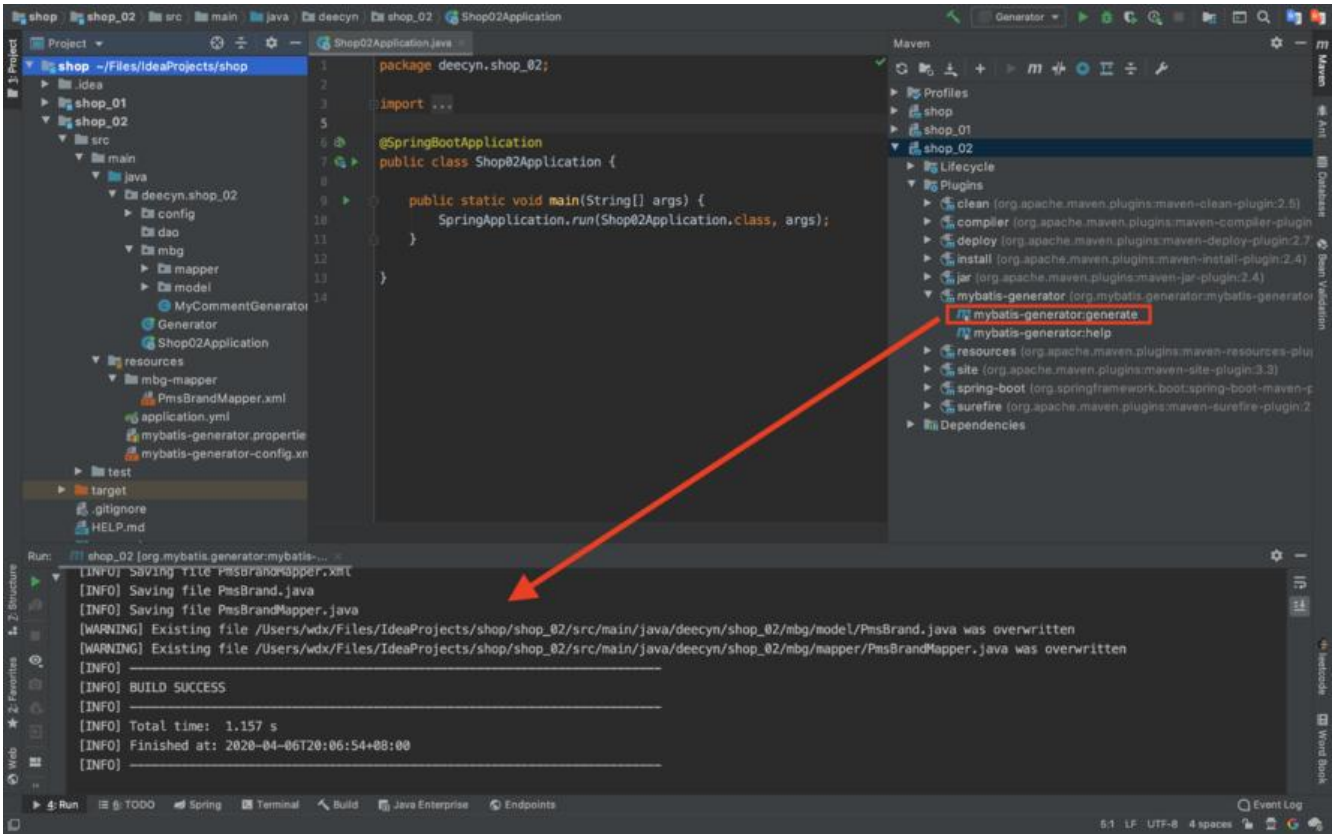
的 pom 文件中引入插件，此外还需要进行一些配置：

```
<build>
  <!-- 为了方便，不展示其它配置... -->
 <plugin>
  <groupId>org.mybatis.generator</groupId>
  <artifactId>mybatis-generator-maven-plugin</artifactId>
  <version>1.3.7</version>

  <configuration>
   <!-- 引入 MyBatis-Generator 的配置文件 -->
   <configurationFile>./src/main/resources/generatorConfig.xml</configurationFile>
   <!-- 允许 MBG 将构建消息写入日志中 -->
   <verbose>true</verbose>
   <!-- 再次运行 MBG 时，允许覆盖已生成的文件，但是不会覆盖 xml 文件 -->
   <overwrite>true</overwrite>
  </configuration>

  <dependencies>
   <!-- 引入 mysql 的 JDBC 驱动，否则会报错找不到驱动 -->
   <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.48</version>
   </dependency>
  </dependencies>
 </plugin>
</build>
```

配置好后，双击 Maven --> Plugins 中的 MyBatis-Generator 运行：



原文链接：一文解析 MyBatis Generator 的使用及配置

即可在相应的目录下生成对应的代码。

需要注意的是，此时，在配置 javaModelGenerator、sqlMapGenerator 和 javaClientGenerator 素的 targetProject 属性时，其路径都是相对于当前 Project 或 Module 的，不需要加前缀。例如当的 Module 名称为 shop_02 时：

```
<javaModelGenerator targetPackage="deecyn.shop_02.mbg.model" targetProject="src/main/
ava"/>
<sqlMapGenerator targetPackage="mbg-mapper" targetProject="src/main/resources"/>
<javaClientGenerator type="XMLMAPPER" targetPackage="deecyn.shop_02.mbg.mapper"
                     targetProject="src/main/java"/>
```

否则会提示找不到对应的 java 和 resource 目录。

四、完整配置文件参考

1. 完整的 pom 配置文件

参考链接：Notes: mybatis-generator-pom

2. 完整的 MyBatis-Generator 配置文件

参考链接：Notes: mybatis-generator-config

五、参考

- 掘金：MyBatis Generator 超详细配置
- 简书：Mybatis Generator完整配置详解
- MyBatis Generator 官网，关于更多配置的详情，建议查看官网的说明。

（完）如有问题，欢迎交流~