MyBatis Geneator详解

作者：abel533(isea533)

博客地址：http://blog.csdn.net/isea533

github:https://github.com/abel533

gitosc:http://git.oschina.net/free

MyBatis Geneator中文文档

MyBatis Geneator中文文档地址：

> http://generator.sturgeon.mopaas.com/

该中文文档由于尽可能和原文内容一致，所以有些地方如果不熟悉，看中文版的文档的也会有一定的障碍，所以本章根据该中文文档以及实际应用，使用通俗的语言来讲解详细的配置。

注:本文后面提到的**MBG**全部指代MyBatis Generator。

运行MyBatis Generator

有4种运行MBG的方法，具体请看文档 运行 MyBatis Generator

MBG下载地址:http://repo1.maven.org/maven2/org/mybatis/generator/mybatis-generator-core/

后续会专门为gitbook完善此部分

XML配置详解

在MBG中，最主要也最重要的就是XML配置文件，因此本篇文章主要的内容就是XML配置。

这里按照配置的顺序对配置逐个讲解，更细的内容可以配合中文文档参照。

1. 配置文件头

```
<?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">
```

使用最新版的MBG需要使用上面的xml头，配置文件必须包含上面的`DOCTYPE`。

2. 根节点`<generatorConfiguration>`

`generatorConfiguration`节点没有任何属性，直接写节点即可，如下：

```
<generatorConfiguration>
    <!-- 具体配置内容 -->
</generatorConfiguration>
```

3. `<generatorConfiguration>`子元素

从这段开始，就是配置的主要内容，这些配置都是`generatorConfiguration`元素的子元素。

包含以下子元素（有严格的顺序）：

1. `<properties>` (0个或1个)
2. `<classPathEntry>` (0个或多个)

3. `<context>` (1个或多个)

3.1 `<properties>` 元素

这个元素用来指定外部的属性元素，不是必须的元素。

元素用于指定一个需要在配置中解析使用的外部属性文件，引入属性文件后，可以在配置中使用`${property}`这种形式的引用，通过这种方式引用属性文件中的属性值。 对于后面需要配置的**jdbc信息**和`targetProject`属性会很有用。

这个属性可以通过`resource`或者`url`来指定属性文件的位置，这两个属性只能使用其中一个来指定，同时出现会报错。

- `resource`：指定**classpath**下的属性文件，使用类似`com/myproject/generatorConfig.properties`这样的属性值。
- `url`：可以指定文件系统上的特定位置，例如`file:///C:/myfolder/generatorConfig.properties`

3.2 `<classPathEntry>` 元素

这个元素可以0或多个，不受限制。

这个元素的作用是将MBG运行时需要用到的jar包(或zip格式)添加到**classpath**下。

最常见的用法是，当**classpath**下面**没有**JDBC驱动的时候，我们通常通过这个属性指定驱动的路径，例如：

```
<classPathEntry location="E:\mysql\mysql-connector-java-5.1.29.jar"/>
```

如果需要用到其他的jar包， 也可以这么配置，例如如果你开发了一个MBG的插件，你就可以通过这种方式加入到**classpath**

这里注意上面重点强调的 **没有**,一般在项目中使用的时候，**classpath**下面都有JDBC驱动，因此从项目中启动的时候不需要配置该项。

建议:由于该参数使用了绝对路径，因此不利用在不同电脑上通用，因此建议最好把需要的jar包放到项目的**classpath**下，避免每个人都得单独配置路径。

3.3 `<context>` 元素

在MBG的配置中，至少需要有一个`<context>`元素。

`<context>`元素用于指定生成一组对象的环境。例如指定要连接的数据库，要生成对象的类型和要处理的数据库中的表。运行MBG的时候还可以指定要运行的`<context>`。

该元素只有一个**必选属性**`id`，用来唯一确定一个`<context>`元素，该`id`属性可以在运行MBG的使用。

此外还有几个**可选属性**：

- `defaultModelType`:**这个属性很重要**，这个属性定义了MBG如何生成**实体类**。
  这个属性有以下可选值：
  - `conditional`:*这是默认值*,这个模型和下面的`hierarchical`类似，除了如果那个单独的类将只包含一个字段，将不会生成一个单独的类。 因此,如果一个表的主键只有一个字段,那么不会为该字段生成单独的实体类,会将该字段合并到基本实体类中。
  - `flat`:该模型为每一张表只生成一个实体类。这个实体类包含表中的所有字段。**这种模型最简单，推荐使用。**
  - `hierarchical`:如果表有主键,那么该模型会产生一个单独的主键实体类,如果表还有BLOB字段， 则会为表生成一个包含所有BLOB字段的单独的实体类,然后为所有其他的字段生成一个单独的实体类。 MBG会在所有生成的实体类之间维护一个继承关系。
- `targetRuntime`:此属性用于指定生成的代码的运行时环境。该属性支持以下可选值：
  - `MyBatis3`:*这是默认值*
  - `MyBatis3Simple`
  - `Ibatis2Java2`
  - `Ibatis2Java5` 一般情况下使用默认值即可，有关这些值的具体作用以及区别请查看中文文档的详细内容。
- `introspectedColumnImpl`:该参数可以指定扩展`org.mybatis.generator.api.IntrospectedColumn`该类的实现类。该属性的作用可以查看扩展 MyBatis Generator。

一般情况下，我们使用如下的配置即可：

```
<context id="Mysql" defaultModelType="flat">
```

如果你希望不生成和`Example`查询有关的内容，那么可以按照如下进行配置:

```
<context id="Mysql" targetRuntime="MyBatis3Simple" defaultModelType="flat">
```

使用`MyBatis3Simple`可以避免在后面的`<table>`中逐个进行配置（后面会提到）。

MBG配置中的其他几个元素，基本上都是`<context>`的子元素，这些子元素（有严格的配置顺序）包括：

- `<property>` (0个或多个)
- `<plugin>` (0个或多个)
- `<commentGenerator>` (0个或1个)
- `<jdbcConnection>` (1个)
- `<javaTypeResolver>` (0个或1个)
- `<javaModelGenerator>` (1个)
- `<sqlMapGenerator>` (0个或1个)
- `<javaClientGenerator>` (0个或1个)
- `<table>` (1个或多个)

其中`<property>`属性比较特殊，后面讲解的时候都会和父元素一起进行讲解。在讲解`<property>`属性前，我们先看看**什么是分隔符？**。

这里通过一个例子说明。假设在Mysql数据库中有一个表名为`user info`，你没有看错，中间是一个空格，这种情况下如果写出`select * from user info`这样的语句，肯定是要报错的，在Mysql中的时候我们一般会写成如下的样子:

```
select * from `user info`
```

这里的使用的**反单引号(`)**就是**分隔符，分隔符**可以用于**表名**或者**列名**。

下面继续看`<property>`支持的属性：

- `autoDelimitKeywords`
- `beginningDelimiter`
- `endingDelimiter`
- `javaFileEncoding`
- `javaFormatter`
- `xmlFormatter`

由于这些属性比较重要，这里一一讲解。

首先是`autoDelimitKeywords`，当表名或者字段名为SQL关键字的时候，可以设置该属性为true，MBG会自动给表名或字段名添加**分隔符**。

然后这里继续上面的例子来讲`beginningDelimiter`和`endingDelimiter`属性。
由于`beginningDelimiter`和`endingDelimiter`的默认值为双引号(")，在Mysql中不能这么写，所以还要将这两个默认值改为**反单引号(`)**，配置如下：

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

属性`javaFileEncoding`设置要使用的Java文件的编码，默认使用当前平台的编码，只有当生产的编码需要特殊指定时才需要使用，一般用不到。

最后两个`javaFormatter`和`xmlFormatter`属性**可能会**很有用，如果你想使用模板来定制生成的java文件和xml文件的样式，你可以通过指定这两个属性的值来实现。

接下来分节对其他的子元素逐个进行介绍。

3.3.1 `<plugin>` 元素

该元素可以配置0个或者多个，不受限制。

`<plugin>`元素用来定义一个插件。插件用于扩展或修改通过MyBatis Generator (MBG)代码生成器生成的代码。

插件将按在配置中配置的顺序执行。

有关插件的详细信息可以参考开发插件和提供的插件了解更多。

3.3.2 `<commentGenerator>` 元素

该元素最多可以配置1个。

这个元素非常有用，相信很多人都有过这样的需求，就是希望MBG生成的代码中可以包含**注释信息**，具体就是生成表或字段的备注信息。

使用这个元素就能很简单的实现我们想要的功能。这里先介绍该元素，介绍完后会举例如何扩展实现该功能。

该元素有一个可选属性`type`,可以指定用户的实现类，该类需要实现`org.mybatis.generator.api.CommentGenerator`接口。而且必有一个默认的构造方法。这个属性接收默认的特殊值`DEFAULT`，会使用默认的实现类`org.mybatis.generator.internal.DefaultCommentGenerator`。

默认的实现类中提供了两个可选属性，需要通过`<property>`属性进行配置。

- `suppressAllComments`:**阻止**生成注释，默认为`false`
- `suppressDate`:**阻止**生成的注释包含时间戳，默认为`false`

一般情况下由于MBG生成的注释信息没有任何价值，而且有时间戳的情况下每次生成的注释都不一样，使用**版本控制**的时候每次都会提交，因而一般情况下我们都会屏蔽注释信息，可以如下配置：

```
<commentGenerator>
    <property name="suppressAllComments" value="true"/>
    <property name="suppressDate" value="true"/>
</commentGenerator>
```

接下来我们简单举例实现生成包含表字段注释信息的注释

因为系统提供了一个默认的实现类，所以对我们来说，自己实现一个会很容易，最简单的方法就是复制默认实现类代码到一个新的文件中，修改类名如`MyCommentGenerator`，在你自己的实现类中，你可以选择是否继续支持上面的两个属性，你还可以增加对其他属性的支持。

我们通过下面一个方法的修改来了解，其他几个方法请自行修改(写本章的时候我也没有完全实现该类，所以不提供完整源码了):

```
@Override
public void addFieldComment(Field field, IntrospectedTable introspectedTable, IntrospectedColumn introspectedColumn) {
    if (introspectedColumn.getRemarks() != null && !introspectedColumn.getRemarks().equals("")) {
        field.addJavaDocLine("/**");
        field.addJavaDocLine(" * " + introspectedColumn.getRemarks());
        addJavadocTag(field, false);
        field.addJavaDocLine(" */");
    }
}
```

这个方法是给字段添加注释信息的，其中`IntrospectedColumn`包含了字段的完整信息，通过`getRemarks`方法可以获取字段的注释信息。上面这个方法修改起来还是很容易的。除了字段的注释外还有`Getter`和`Setter`，以及类的注释。此外还有生成XML的注释，大家可以根据默认的实现进行修改。

完成我们自己的实现类后，我们还需要做如下配置：

```
<commentGenerator type="com.github.abel533.mybatis.generator.MyCommentGenerator"/>
```

3.3.3 `<jdbcConnection>` 元素

`<jdbcConnection>`用于指定数据库连接信息，该元素必选，并且只能有一个。

配置该元素只需要注意如果JDBC驱动不在**classpath**下，就需要通过`<classPathEntry>`元素引入jar包，这里**推荐将jar包放到classpath**下。

该元素有两个必选属性:

- `driverClass`:访问数据库的JDBC驱动程序的完全限定类名
- `connectionURL`:访问数据库的JDBC连接URL

该元素还有两个可选属性:

- `userId`:访问数据库的用户ID
- `password`:访问数据库的密码

此外该元素还可以接受多个`<property>`子元素，这里配置的`<property>`属性都会添加到JDBC驱动的属性中。

这个元素配置起来最容易，这里举个简单例子：

```
<jdbcConnection driverClass="com.mysql.jdbc.Driver"
                connectionURL="jdbc:mysql://localhost:3306/test"
                userId="root"
                password="">
</jdbcConnection>
```

3.3.4 `<javaTypeResolver>` 元素

该元素最多可以配置一个。

这个元素的配置用来指定JDBC类型和Java类型如何转换。

该元素提供了一个可选的属性`type`，和`<commentGenerator>`比较类型，提供了默认的实现`DEFAULT`，一般情况下使用默认即可，需要特殊处理的情况可以通过其他元素配置来解决，不建议修改该属性。

该属性还有一个可以配置的`<property>`元素。

可以配置的属性为`forceBigDecimals`，该属性可以控制是否强制`DECIMAL`和`NUMERIC`类型的字段转换为Java类型的`java.math.BigDecimal`,默认值为`false`，一般不需要配置。

默认情况下的转换规则为：

1. 如果`精度>0`或者`长度>18`，就会使用`java.math.BigDecimal`
2. 如果`精度=0`并且`10<=长度<=18`，就会使用`java.lang.Long`
3. 如果`精度=0`并且`5<=长度<=9`，就会使用`java.lang.Integer`
4. 如果`精度=0`并且`长度<5`，就会使用`java.lang.Short`

如果设置为`true`，那么一定会使用`java.math.BigDecimal`，配置示例如下：

```
<javaTypeResolver >
    <property name="forceBigDecimals" value="true" />
</javaTypeResolver>
```

3.3.5 `<javaModelGenerator>` 元素

该元素必须配置一个，并且最多一个。

该元素用来控制生成的实体类，根据`<context>`中配置的`defaultModelType`，一个表可能会对应生成多个不同的实体类。一个表对应多个类实际上并不方便，所以前面也推荐使用`flat`，这种情况下一个表对应一个实体类。

该元素只有两个属性，都是必选的。

- `targetPackage`:生成实体类存放的包名，一般就是放在该包下。实际还会受到其他配置的影响(`<table>`中会提到)。
- `targetProject`:指定目标项目路径，使用的是文件系统的绝对路径。

该元素支持以下几个`<property>`子元素属性：

- `constructorBased`:该属性只对`MyBatis3`有效，如果`true`就会使用构造方法入参，如果`false`就会使用`setter`方式。默认为`false`。
- `enableSubPackages`:如果`true`，MBG会根据`catalog`和`schema`来生成子包。如果`false`就会直接用`targetPackage`属性。默认为`false`。
- `immutable`:该属性用来配置实体类属性是否可变，如果设置为`true`，那么`constructorBased`不管设置成什么，都会使用构造方法入参，并且不会生成`setter`方法。如果为`false`，实体类属性就可以改变。默认为`false`。
- `rootClass`:设置所有实体类的基类。如果设置，需要使用类的全限定名称。并且如果MBG能够加载`rootClass`，那么MBG不会覆盖和父类中完全匹配的属性。匹配规则：
    - 属性名完全相同
    - 属性类型相同
    - 属性有`getter`方法
    - 属性有`setter`方法
- `trimStrings`:是否对数据库查询结果进行`trim`操作，如果设置为`true`就会生成类似这样`public void setUsername(String username) {this.username = username == null ? null : username.trim();}`的`setter`方法。默认值为`false`。

配置示例如下：

```
<javaModelGenerator targetPackage="test.model" targetProject="E:\MyProject\src\main\java">
    <property name="enableSubPackages" value="true" />
    <property name="trimStrings" value="true" />
</javaModelGenerator>
```

3.3.6 `<sqlMapGenerator>` 元素

该元素可选，最多配置一个。但是有如下两种必选的特殊情况：

- 如果`targetRuntime`目标是**iBATIS2**，该元素必须配置一个。
- 如果`targetRuntime`目标是**MyBatis3**，只有当`<javaClientGenerator>`需要XML时，该元素必须配置一个。 如果没有配置`<javaClientGenerator>`，则使用以下的规则：
  - 如果指定了一个`<sqlMapGenerator>`，那么MBG将只生成XML的SQL映射文件和实体类。
  - 如果没有指定`<sqlMapGenerator>`，那么MBG将只生成实体类。

该元素只有两个属性（和前面提过的`<javaModelGenerator>`的属性含义一样），都是必选的。

- `targetPackage`:生成实体类存放的包名，一般就是放在该包下。实际还会受到其他配置的影响(`<table>`中会提到)。
- `targetProject`:指定目标项目路径，使用的是文件系统的绝对路径。

该元素支持`<property>`子元素，只有一个可以配置的属性：

- `enableSubPackages`:如果`true`，MBG会根据`catalog`和`schema`来生成子包。如果`false`就会直接用`targetPackage`属性。默认为`false`。

配置示例：

```
<sqlMapGenerator targetPackage="test.xml"  targetProject="E:\MyProject\src\main\resources">
    <property name="enableSubPackages" value="true" />
</sqlMapGenerator>
```

3.3.7 `<javaClientGenerator>` 元素

该元素可选，最多配置一个。

如果不配置该元素，就不会生成Mapper接口。

该元素有3个必选属性：

- `type`:该属性用于选择一个预定义的客户端代码（可以理解为Mapper接口）生成器，用户可以自定义实现，需要继承`org.mybatis.generator.codegen.AbstractJavaClientGenerator`类，必选有一个默认的构造方法。 该属性提供了以下预定的代码生成器，首先根据`<context>`的`targetRuntime`分成三类：
  - **MyBatis3**:
    - **ANNOTATEDMAPPER**:基于注解的Mapper接口，不会有对应的XML映射文件
    - **MIXEDMAPPER**:XML和注解的混合形式，(上面这种情况中的)`SqlProvider`注解方法会被XML替代。
    - **XMLMAPPER**:所有的方法都在XML中，接口调用依赖XML文件。
  - **MyBatis3Simple**:
    - **ANNOTATEDMAPPER**:基于注解的Mapper接口，不会有对应的XML映射文件
    - **XMLMAPPER**:所有的方法都在XML中，接口调用依赖XML文件。
  - **Ibatis2Java2**或**Ibatis2Java5**:
    - **IBATIS**:生成的对象符合iBATIS的DAO框架（不建议使用）。
    - **GENERIC-CI**:生成的对象将只依赖于SqlMapClient，通过构造方法注入。
    - **GENERIC-SI**:生成的对象将只依赖于SqlMapClient，通过`setter`方法注入。
    - **SPRING**:生成的对象符合Spring的DAO接口
- `targetPackage`:生成实体类存放的包名，一般就是放在该包下。实际还会受到其他配置的影响(`<table>`中会提到)。
- `targetProject`:指定目标项目路径，使用的是文件系统的绝对路径。

该元素还有一个可选属性：

- `implementationPackage`:如果指定了该属性，实现类就会生成在这个包中。

该元素支持`<property>`子元素设置的属性：

- `enableSubPackages`
- `exampleMethodVisibility`
- `methodNameCalculator`
- `rootInterface`
- `useLegacyBuilder`

这几个属性不太常用，具体作用请看完整的文档，这里对`rootInterface`做个简单介绍。

`rootInterface`用于指定一个所有生成的接口都继承的父接口。 这个值可以通过`<table>`配置的`rootInterface`属性覆盖。

这个属性对于通用Mapper来说，可以让生成的所有接口都继承该接口。

配置示例：

```
<javaClientGenerator type="XMLMAPPER" targetPackage="test.dao"
             targetProject="E:\MyProject\src\main\java"/>
```

3.3.8 `<table>` 元素

该元素至少要配置一个，可以配置多个。

该元素用来配置要通过内省的表。只有配置的才会生成实体类和其他文件。

该元素有一个必选属性：

- `tableName`：指定要生成的表名，可以使用SQL通配符匹配多个表。

例如要生成全部的表，可以按如下配置：

```
<table tableName="%" />
```

该元素包含多个可选属性：

- `schema`:数据库的schema,可以使用SQL通配符匹配。如果设置了该值，生成SQL的表名会变成如`schema.tableName`的形式。
- `catalog`:数据库的catalog，如果设置了该值，生成SQL的表名会变成如`catalog.tableName`的形式。
- `alias`:如果指定，这个值会用在生成的select查询SQL的表的别名和列名上。 列名会被别名为 alias*actualColumnName(别名*实际列名) 这种模式。
- `domainObjectName`:生成对象的基本名称。如果没有指定，MBG会自动根据表名来生成名称。
- `enableXXX`:XXX代表多种SQL方法，该属性用来指定是否生成对应的XXX语句。
- `selectByPrimaryKeyQueryId`:DBA跟踪工具会用到，具体请看详细文档。
- `selectByExampleQueryId`:DBA跟踪工具会用到，具体请看详细文档。
- `modelType`:和`<context>`的`defaultModelType`含义一样，这里可以针对表进行配置，这里的配置会覆盖`<context>`的`defaultModelType`配置。
- `escapeWildcards`:这个属性表示当查询列，是否对schema和表名中的SQL通配符 (*'' and '%') 进行转义。 对于某些驱动当schema或表名中包含SQL通配符时（例如，一个表名是MY*TABLE，有一些驱动需要将下划线进行转义）是必须的。默认值是`false`。
- `delimitIdentifiers`:是否给标识符增加**分隔符**。默认`false`。当`catalog`,`schema`或`tableName`中包含空白时，默认为`true`。
- `delimitAllColumns`:是否对所有列添加**分隔符**。默认`false`。

该元素包含多个可用的`<property>`子元素，可选属性为：

- `constructorBased`:和`<javaModelGenerator>`中的属性含义一样。
- `ignoreQualifiersAtRuntime`:生成的SQL中的表名将不会包含`schema`和`catalog`前缀。
- `immutable`:和`<javaModelGenerator>`中的属性含义一样。
- `modelOnly`:此属性用于配置是否为表只生成实体类。如果设置为`true`就不会有Mapper接口。如果配置了`<sqlMapGenerator>`，并且`modelOnly`为`true`，那么XML映射文件中只有实体对象的映射元素(`<resultMap>`)。如果为`true`还会覆盖属性中的`enableXXX`方法，将不会生成任何CRUD方法。
- `rootClass`:和`<javaModelGenerator>`中的属性含义一样。
- `rootInterface`:和`<javaClientGenerator>`中的属性含义一样。
- `runtimeCatalog`:运行时的`catalog`，当生成表和运行环境的表的`catalog`不一样的时候可以使用该属性进行配置。
- `runtimeSchema`:运行时的`schema`，当生成表和运行环境的表的`schema`不一样的时候可以使用该属性进行配置。
- `runtimeTableName`:运行时的`tableName`，当生成表和运行环境的表的`tableName`不一样的时候可以使用该属性进行配置。
- `selectAllOrderByClause`:该属性值会追加到`selectAll`方法后的SQL中，会直接跟`order by`拼接后添加到SQL末尾。
- `useActualColumnNames`:如果设置为true,那么MBG会使用从数据库元数据获取的列名作为生成的实体对象的属性。 如果为false(默认值)，MGB将

会尝试将返回的名称转换为驼峰形式。 在这两种情况下，可以通过 元素显示指定，在这种情况下将会忽略这个（useActualColumnNames）属性。

- `useColumnIndexes`:如果是true,MBG生成resultMaps的时候会使用列的索引,而不是结果中列名的顺序。
- `useCompoundPropertyNames`:如果是true,那么MBG生成属性名的时候会将列名和列备注接起来. 这对于那些通过第四代语言自动生成列(例如:FLD22237),但是备注包含有用信息(例如:"customer id")的数据库来说很有用. 在这种情况下,MBG会生成属性名FLD2237_CustomerId。

除了`<property>`子元素外，`<table>`还包含以下子元素：

- `<generatedKey>` (0个或1个)
- `<columnRenamingRule>` (0个或1个)
- `<columnOverride>` (0个或多个)
- `<ignoreColumn>` (0个或多个)

下面对这4个元素进行详细讲解。

1. `<generatedKey>` 元素

这个元素最多可以配置一个。
这个元素用来指定自动生成主键的属性（identity字段或者sequences序列）。如果指定这个元素，MBG在生成insert的SQL映射文件中插入一个`<selectKey>`元素。 这个元素**非常重要**，这个元素包含下面两个必选属性：

- `column`:生成列的列名。
- `sqlStatement`:将返回新值的 SQL 语句。如果这是一个identity列，您可以使用其中一个预定义的的特殊值。预定义值如下：
  - **Cloudscape**
  - **DB2**
  - **DB2_MF**
  - **Derby**
  - **HSQLDB**
  - **Informix**
  - **MySql**
  - **SqlServer**
  - **SYBASE**
  - **JDBC**:这会配置MBG使用MyBatis3支持的JDBC标准的生成key来生成代码。 这是一个独立于数据库获取标识列中的值的方法。 **重要**: 只有当目标运行为MyBatis3时才会产生正确的代码。 如果与iBATIS2一起使用目标运行时会产生运行时错误的代码。

这个元素还包含两个可选属性：

- `identity`:当设置为`true`时,该列会被标记为`identity`列， 并且`<selectKey>`元素会被插入在`insert`后面。 当设置为`false`时， `<selectKey>`会插入到`insert`之前（通常是序列）。**重要**: 即使您`type`属性指定为`post`，您仍然需要为`identity`列将该参数设置为`true`。 这将标志MBG从插入列表中删除该列。默认值是`false`。
- `type`:`type=post and identity=true`的时候生成的`<selectKey>`中的`order=AFTER`,当`type=pre`的时候，`identity只能为false`，生成的`<selectKey>`中的`order=BEFORE`。可以这么理解，自动增长的列只有插入到数据库后才能得到ID，所以是`AFTER`,使用序列时，只有先获取序列之后，才能插入数据库，所以是`BEFORE`。

配置示例一：

```
<table tableName="user login info" domainObjectName="UserLoginInfo">
    <generatedKey column="id" sqlStatement="Mysql"/>
</table>
```

对应的生成的结果：

```
<insert id="insert" parameterType="test.model.UserLoginInfo">
    <selectKey keyProperty="id" order="BEFORE" resultType="java.lang.Integer">
        SELECT LAST_INSERT_ID()
    </selectKey>
    insert into `user login info` (Id, username, logindate, loginip)
    values (#{id,jdbcType=INTEGER}, #{username,jdbcType=VARCHAR}, #{logindate,jdbcType=TIMESTAMP}, #{loginip,jdbcType=VARCHAR})
</insert>
```

配置示例二：

```
<table tableName="user login info" domainObjectName="UserLoginInfo">
    <generatedKey column="id" sqlStatement="select SEQ_ID.nextval from dual"/>
</table>
```

对应的生成结果：

```
<insert id="insert" parameterType="test.model.UserLoginInfo">
    <selectKey keyProperty="id" order="BEFORE" resultType="java.lang.Integer">
        select SEQ_ID.nextval from dual
    </selectKey>
    insert into `user login info` (Id, username, logindate, loginip)
    values (#{id,jdbcType=INTEGER}, #{username,jdbcType=VARCHAR}, #{logindate,jdbcType=TIMESTAMP},#{loginip,jdbcType=VARCHAR})
</insert>
```

配置示例三：

```
<table tableName="user login info" domainObjectName="UserLoginInfo">
    <generatedKey column="id" sqlStatement="JDBC"/>
</table>
```

对应的生成结果：

```
  <insert id="insert" keyProperty="id" parameterType="test.model.UserLoginInfo" useGeneratedKeys="true">
        insert into user login info (username, logindate, loginip)
        values (#{username,jdbcType=VARCHAR}, #{logindate,jdbcType=TIMESTAMP}, #{loginip,jdbcType=VARCHAR})
  </insert>
```

2. `<columnRenamingRule>` 元素

该元素最多可以配置一个，使用该元素可以在生成列之前，对列进行重命名。这对那些存在同一前缀的字段想在生成属性名时去除前缀的表非常有用。 例如假设一个表包含以下的列：

- `CUST_BUSINESS_NAME`
- `CUST_STREET_ADDRESS`
- `CUST_CITY`
- `CUST_STATE`

生成的所有属性名中如果都包含`CUST`的前缀可能会让人不爽。这些前缀可以通过如下方式定义重命名规则:

```
<columnRenamingRule searchString="^CUST_" replaceString="" />
```

注意，在内部，MBG使用`java.util.regex.Matcher.replaceAll`方法实现这个功能。 请参阅有关该方法的文档和在Java中使用正则表达式的例子。

当`<columnOverride>`匹配一列时，这个元素（`<columnRenamingRule>`）会被忽略。`<columnOverride>`优先于重命名的规则。

该元素有一个必选属性：

- `searchString`:定义将被替换的字符串的正则表达式。

该元素有一个可选属性：

- `replaceString`:这是一个用来替换搜索字符串列每一个匹配项的字符串。如果没有指定，就会使用空字符串。

关于`<table>`的`<property>`属性`useActualColumnNames`对此的影响可以查看完整文档。

3. `<columnOverride>` 元素

该元素可选，可以配置多个。

该元素从将某些属性默认计算的值更改为指定的值。

该元素有一个必选属性:

- `column`:要重写的列名。

该元素有多个可选属性：

- `property`:要使用的Java属性的名称。如果没有指定，MBG会根据列名生成。 例如，如果一个表的一列名为`STRT_DTE`，MBG会根据`<table>`的`useActualColumnNames`属性生成`STRT_DTE`或`strtDte`。
- `javaType`:该列属性值为完全限定的Java类型。如果需要，这可以覆盖由`JavaTypeResolver`计算出的类型。 对某些数据库来说，这是必要的用来处理**“奇怪的”**数据库类型（例如MySql的`unsigned bigint`类型需要映射为`java.lang.Object`）。
- `jdbcType`:该列的JDBC类型(`INTEGER, DECIMAL, NUMERIC, VARCHAR`等等)。 如果需要，这可以覆盖由`JavaTypeResolver`计算出的类型。 对某些数据库来说，这是必要的用来处理怪异的JDBC驱动 (例如DB2的`LONGVARCHAR`类型需要为iBATIS 映射为`VARCHAR`)。
- `typeHandler`:用户定义的需要用来处理这列的类型处理器。它必须是一个继承iBATIS的`TypeHandler`类或`TypeHandlerCallback`接口（该接口很容易继承）的全限定的类名。如果没有指定或者是空白，iBATIS会用默认的类型处理器来处理类型。**重要**:MBG不会校验这个类型处理器是否存在或者可用。 MGB只是简单的将这个值插入到生成的SQL映射的配置文件中。
- `delimitedColumnName`:指定是否应在生成的SQL的列名称上增加**分隔符**。 如果列的名称中包含空格，MGB会自动添加**分隔符**， 所以这个重写只有当列名需要强制为一个合适的名字或者列名是数据库中的保留字时是必要的。

配置示例：

```
<table schema="DB2ADMIN" tableName="ALLTYPES" >
    <columnOverride column="LONG_VARCHAR_FIELD" javaType="java.lang.String" jdbcType="VARCHAR" />
</table>
```

4. `<ignoreColumn>` 元素

该元素可选，可以配置多个。

该元素可以用来屏蔽不需要生成的列。

该元素有一个必选属性：

- `column`:要忽略的列名。

该元素还有一个可选属性：

- `delimitedColumnName`:匹配列名的时候是否区分大小写。如果为`true`则区分。默认值为`false`，不区分大小写。

MyBatis Generator最佳实践

本节内容针对MyBatis3，使用iBATIS的不一定适用。

以下根据个人经验（对此有意见的可以留言）对一些配置看法列出如下几点：

1. 关于`Example`方法，`Example`方法虽然很强大，但是SQL不易管理，因此不建议使用。
2. 取消`Example`方法的配置，通过`<table>`上的`enablexxExample`方法可以屏蔽，但是最好的方法是在`<context>`上设置`targetRuntime="MyBatis3Simple"`。
3. 关于实体类的`modelType`，建议使用`defaultModelType="flat"`，只有一个对象的情况下管理毕竟方便，使用也简单。
4. 关于注释`<commentGenerator>`，不管你是否要重写自己的注释生成器，有一点不能忘记，那就是注释中一定要保留`@mbggenerated`,MBG通过该字符串来判断代码是否为代码生成器生成的代码，有该标记的的代码在重新生成的时候会被删除，不会重复。不会在XML中出现重复元素。
5. 使用MBG生成的代码时，建议尽可能不要去修改自动生成的代码，而且要生成带有`@mbggenerated`，这样才不会在每次重新生成代码的时候需要手动修改好多内容。
6. 仍然是注释相关，在`<commentGenerator>`中，建议一定要保留`suppressAllComments`属性(使用默认值`false`)，一定要取消(设为`true`)时间戳`suppressDate`，避免重复提交SVN。
7. `<jdbcConnection>`建议将JDBC驱动放到项目的**classpath**下，而不是使用`<classPathEntry>`来引入jar包，主要考虑到所有开发人员的统一性。
8. 当数据库字段使用`CHAR`时，建议在`<javaModelGenerator>`中设置`<property name="trimStrings" value="true" />`，可以自动去掉不必要的空格。
9. 在`<javaClientGenerator>`中，建议设置`type="XMLMAPPER"`,不建议使用注解或混合模式，比较代码和SQL完全分离易于维护。
10. 建议尽可能在`<table>`中配置`<generatedKey>`，避免手工操作，以便于MBG重复执行代码生成。

如果有其他有价值的经验，会继续补充。

综合以上信息，这里给出一个`Mysql`的简单配置：

```
<?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>
    <context id="MysqlContext" targetRuntime="MyBatis3Simple" defaultModelType="flat">
        <property name="beginningDelimiter" value="`"/>
        <property name="endingDelimiter" value="`"/>

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

        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                        connectionURL="jdbc:mysql://localhost:3306/test"
                        userId="root"
                        password="">
        </jdbcConnection>

        <javaModelGenerator targetPackage="test.model" targetProject="G:\MyProject\src\main\java">
            <property name="trimStrings" value="true" />
        </javaModelGenerator>

        <sqlMapGenerator targetPackage="test.xml"  targetProject="G:\MyProject\src\main\resources"/>

        <javaClientGenerator type="XMLMAPPER" targetPackage="test.dao"  targetProject="G:\MyProject\src\main\java"/>

        <table tableName="%">
            <generatedKey column="id" sqlStatement="Mysql"/>
        </table>
    </context>
</generatorConfiguration>
```

`<table>`这里用的通配符匹配全部的表，另外所有表都有自动增长的`id`字段。如果不是所有表的配置都一样，可以做针对性的配置。