**MyBatis Generator介绍**

MyBatis Generator (MBG) 是一个Mybatis的代码生成器 [MyBatis](http://mybatis.org) 和 [iBATIS](http://ibatis.apache.org). 他可以生成Mybatis各个版本的代码，和iBATIS 2.2.0版本以后的代码。 他可以内省数据库的表（或多个表）然后生成可以用来访问（多个）表的基础对象。 这样和数据库表进行交互时不需要创建对象和配置文件。 MBG的解决了对数据库操作有最大影响的一些简单的CRUD（插入，查询，更新，删除）操作。 您仍然需要对联合查询和存储过程手写SQL和对象。

MyBatis Generator 会生成:

* 匹配表结构的Java POJO，可能包括:
  + 一个和表主键匹配的类(如果存在主键[注:只有联合主键会有])
  + 一个包含了非主键字段的类(BLOB字段除外[注:单字段做主键时这里会包含])
  + 一个包含了BLOB字段的类 (如果表包含了BLOB字段)
  + 一个允许动态查询、更新和删除的类[注:指的是Example查询]

这些类之间会有适当的继承关系。 请注意可以配置生成器来生成不同类型的 POJO 的层次结构。 例如，如果您愿意您可能会选择针对每个表生成一个单独的实体对象。

* MyBatis/iBATIS 兼容 SQL 映射 XML 文件。MBG 在配置中为每个表简单的 CRUD 操作生成 SQL。 生成的 SQL 语句包括：
  + insert (插入)
  + update by primary key (根据主键更新记录)
  + update by example (根据条件更新记录)
  + delete by primary key (根据主键删除记录)
  + delete by example (根据条件删除记录)
  + select by primary key (根据主键查询记录)
  + select by example (根据条件查询记录集)
  + count by example (根据条件查询记录总数)

根据表的结构，生成的这些语句会有不同的变化（例如，如果表中没有主键，那么 MBG 将不会生成update by primary key方法）。

* Java客户端类会适当的使用上面的对象，生成Java客户端类时可选的。 MBG会为MyBatis 3.x生成如下客户端类：
  + 一个可以和MyBatis 3.x一起使用的mapper接口类

MBG会为iBATIS 2.x生成如下的客户端类：

* + 符合 [Spring](http://www.springframework.org) 框架的DAO类。
  + 只使用iBATIS SQL映射API的DAO。 这种DAO可以通过下面两种方式生成：提供SqlMapClient通过构造方法或者setter注入。
  + 符合iBATIS DAO 框架的DAO (iBATIS可选的一部分，这一框架已经过时，我们建议您使用Spring框架替代)。

MyBatis generator 可以在迭代开发环境中良好的运行， 在持续的构建环境中作为一个ant任务或maven插件。 运行MBG时要记住以下重要的事：

1. MBG 会自动合并已经存在并且和新生成的文件重名的 XML。MBG 不会覆盖您对已经生成xml所做的修改。 您可以反复的运行而不必担心失去您自定义的更改。 MBG 将取代所有以前运行中生成的 XML 元素。
2. MBG **不会** 合并 Java 文件,他可以覆盖已经存在的文件或者保存新生成的文件为一个不同的唯一的名字。 您可以手动合并这些更改。 当您使用[Eclipse](http://www.eclipse.org) 插件时, MBG 可以自动合并 Java 文件.

**依赖项**

MBG依赖JRE,需要JRE6.0或以上版本。 此外，还有一个继承了DatabaseMetaData接口的JDBC驱动。 特别是getColumns和getPrimaryKeys两个方法时必须的。

**支持**

通过Mybatis用户邮件列表提供对Mybatis generator的支持。 您可以通过谷歌订阅或查看邮件列表的代码:

<http://groups.google.com/group/mybatis-user>

如果您认为您找到了一个 bug，在您创建新的Issue前，请在用户列表中询问一下。 如果您发现一个 bug，或者有新的功能要求，您可以在github上打开一个新的Issue：

<https://github.com/mybatis/generator/issues>

**译者**

译者水平有限，如果您发现翻译不通顺的地方，您可以在下面的地址创建Issue：

<http://git.oschina.net/free/mybatis-generator-core/issues>

**译者列表**

* abel533(isea533)
  + github:<https://github.com/abel533>
  + gitosc:<http://git.oschina.net/free>
  + blog:<http://blog.csdn.net/isea533>
* hs598375774sa
  + oschina:<http://my.oschina.net/hssa>
* ' ╃锦色ㄨ年华²º¹⁴

**MyBatis Generator新增功能**

**Version 1.3.3**

**公告**

* MyBatis Generator 需要 JRE 1.6 或更高版本的支持

**改进**

* 从 MyBatis 3.2 开始使用了新的 SQL 生成器类。 如果您必须在 MyBatis 旧版本上运行，在您的 < JavaClientGenerator > 配置上设置属性"useLegacyBuilder"为"true"。
* Issue #11 - maven插件现在会输出到maven日志 (thanks to Paul Krause)
* Issue #4 - 如果任何列将解析为一个 Java 保留字，生成器现在会抛出一个警告

**Bugs 修复**

* Fixed issue #648 - Missing imports when using constructor based models
* Issues #10 - EqualsHashCodePlugin now uses the java.util.Arrays methods for array attributes.

**Version 1.3.2**

**Bugs 修复**

* Add comments to generated constructors in the model classes so they can be merged.
* Support package names with Uppercase elements.
* Fixed issue #288 - Incorrect annotated countByExample method
* Fixed the Maven plugin so <properties> files can be found in the project classpath.
* Fixed issue #359 - make JdbcTypeInformation public
* Fixed Context.toXmlElement() method to include missing attributes
* Fixed CaseInsensitiveLikePlugin to add new methods to GeneratedCriteria inner class
* Issue #412 - update the documentation to reflect the difference in MyBatis3 regarding generated keys.
* Issue #440 - incorrect code generated for primitives with type handlers
* Issue #439 - use auto boxing for primitives when appropriate
* Issue #438 - keep primary key properties in database sequence
* Issue #507 - RowBounds plugin generates duplicate statements
* Issue #593 - CaseInsensitiveLikePlugin skipped Jdbc4 National Character Types

**改进**

* Added a new target runtime - MyBatis3Simple - that can be used to generate very simple CRUD operations on tables. This runtime produces far simpler MyBatis code than the normal MyBatis3 runtime. The produced code also has lower functionality that the normal MyBatis3 runtime. If you do not regularly use the "by example" methods, then the code produced by the MyBatis3Siple runtime may be more suitable to your project.
* Added a new plugin - VirtualPrimaryKey plugin - that can be used to specify columns that act as a primary key, even if they are not defined as primary keys in the database.
* Issue #328 - Added a new plugin - RowBounds plugin - that will generate an additional version of selectByExample that supports the MyBatis RowBounds function.
* Created a new reference page for supplied plugins: [Supplied Plugins](http://mbg.cndocs.tk/reference/plugins.html)
* Issue #368 - Allowed for generation of Java Model only. If no SQLMapGenerator or JavaClientGenerator is specified in a context, then only a Java Model is generated. Also, if property "modelOnly" is set to "true" on a <table> element, then only model object and, possibly XML result maps, will be generated.
* Issue #374 - Allow for specification of a file encoding for reading/writing Java files on the file system. There is a new property "javaFileEncoding" on the <context> element that can be used to specify a Java file encoding. (XML files are always read/written in UTF-8 per the spec).
* Added the ability to specify custom code formatters for generated Java and XML files. See new properties on the <context> element for more information.
* Added support for varargs to org.mybatis.generator.api.dom.java.Parameter class
* Added support for synchronized and native to org.mybatis.generator.api.dom.java.Method class
* Added support for transient and volatile to org.mybatis.generator.api.dom.java.Field class
* Issue #375 - Added a new plugin - ToStringPlugin - that will generate a toString() method in the model classes. Thanks Alexei and Iwao!
* Issue #233 - added GWT capability to the Serializable plugin
* Issue #564 - support subpackages at the table level
* Issue #590 - new plugin for <cache>. Thanks Jason Bennett!

**Version 1.3.1**

**Bugs 修复**

* Always specify <selectKey> order for MyBatis3 - position within the <insert> is not relevant.
* "rootInterface" was ignored for XMLMAPPER clients
* Fixed bug when a lower case class name is specified as a domain object name
* Fixed issue #174 - incorrect format of order by clause in selectByExampleWithBlobs

**改进**

* Added a new MyBatis3 generator that generates code based solely on annotations with no generated XML. Configuration settings for this new generator are as follows:
  + The targetRuntime attribute of [<context>](http://mbg.cndocs.tk/configreference/context.html) element is **MyBatis3**
  + The type attribute of [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) element is **ANNOTATEDMAPPER**
* Added a new MyBatis3 generator that generates code based on a mix of annotations and generated XML. Configuration settings for this new generator are as follows:
  + The targetRuntime attribute of [<context>](http://mbg.cndocs.tk/configreference/context.html) element is **MyBatis3**
  + The type attribute of [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) element is **MIXEDMAPPER**
* Add support for JDBC types NCHAR, NCLOB, NVARCHAR to match MyBatis3.
* Added support for "useGeneratedKeys" in MyBatis3. See [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html) for details.
* Added support for immutable objects and constructor based result maps in MyBatis3. See [<table>](http://mbg.cndocs.tk/configreference/table.html) and/or [<javaModelGenerator>](http://mbg.cndocs.tk/configreference/javaModelGenerator.html) for details.
* Added support for initialization blocks in the Java DOM
* Issue #214 - Added the ability to supress all comments in generated code. See [<commentGenerator>](http://mbg.cndocs.tk/configreference/commentGenerator.html) for details.

**Version 1.3.0**

1. Moved to mybatis.org, renamed MyBatis Generator
2. MyBatis generator will continue to support XML configuration files from Ibator. However, any new features will only be implemented in MyBatis formatted configuration files. Some minimal changes are required to migrate Ibator formatted configuration files to the new DTD for MyBatis Generator.
3. Configuration settings for MyBatis 3 are as follows:
   1. The targetRuntime attribute of [<context>](http://mbg.cndocs.tk/configreference/context.html) element must be changed to **MyBatis3**
   2. The type attribute of [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) element must be changed to **XMLMAPPER**
4. Removed support for suppressTypeWarnings property in the <context> element. This confusing property has become unmanagable due to confusion between @SuppressWarnings("unchecked") and @SuppressWarnings("rawtypes") in the different compilers. This was only used in the corner case where code generated with Ibatis2Java2 targetRuntime was to be compiled with a JSE 5.0 compiler.

**Version 1.2.2 (Never Released)**

**Announcements**

* The org.apache.ibatis.ibator.api.CommentGenerator interface has changed. Classes that implement this interface **must be changed**. With this change, implementing classes have access to many more data elements from which to generate comments. Additionally, this change makes the comment generator interface more consistent with the other public Ibator interfaces. Details of the change follow:
  + Methods which accepted the parameter FullyQualifiedTable now accept the parameter IntrospectedTable instead. The FullyQualifiedTable instance is available through the method IntrospectedTable.getFullyQualifiedTable().
  + Methods which accepted the String parameter columnName now accept the parameter IntrospectedColumn instead. The column name is available through the method IntrospectedColumn.getActualColumnName().

**重要提示:** any implementation that subclasses the Ibator supplied class DefaultCommentGenerator does not need to change immediately with this release. The old methods have been deprecated and will be removed with the next release of Ibator - so subclasses should be reworked as soon as convenient.

* The SQL Map generator has changed in that it no longer prepends the string "ibatorgenerated\_" to every generated XML id. If you were relying on these generated names in other code, you can force Ibator to prepend the string with the property useLegacyXMLIds on the [<SqlMapGenerator>](http://mbg.cndocs.tk/configreference/sqlMapGenerator.html) configuration.
* Ibator is now built with Maven and includes a Maven plugin

**Bugs 修复**

* NPE when no DAOs are generated.
* IBATIS-579 - Don't allow column names that contain spaces to break across lines in generated XML.
* Fixed NPE and incorrect calculation in generated equals method (from EqualsHashCodePlugin) when certain fields are null - thanks to Benjamin Klatt for finding this bug.
* IBATIS-601 - improper validation of <generatedKey>
* IBATIS-609 - incorrect parsing of Java generic types
* Fixed spelling error LONCVARCHAR to LONGVARCHAR (thanks Allard)
* Fixed IBATIS-731 - change name of primary key variable to avoid conflicts
* Fixed IBATIS-699 - Overwrite unmergeable XML files if enabled
* Fixed issue where insertSelective failed if there is a sequence generating the primary key (issue only with iBATIS3)

**改进**

* IBATIS-569 - Modified the IbatorRules implementation to make it easier for plugins to provide custom implementations of IbatorRules. See the Javadoc for the new class org.apache.ibatis.ibator.internal.rules.IbatorRulesDelegate for more information.
* IBATIS-571 - Added support for automatically delimiting SQL keywords if they are used as column names in tables. See the [<ibatorContext>](http://mbg.cndocs.tk/configreference/ibatorContext.html) page for more information.
* IBATIS-577 - Define SQL fragments for column lists to improve reusability of generated code. Thanks to Iwao AVE! for the idea and the initial patch.
* Added new -verbose command line argument. See the [Running Ibator](http://mbg.cndocs.tk/running.html) page for more information.
* Added logging statements for use in debug. See the [Logging](http://mbg.cndocs.tk/reference/logging.html) page for more information.
* Added new example plugin to demonstrate adding case insensitive LIKE support to generated Example classes. See the [<ibatorPlugin>](http://mbg.cndocs.tk/configreference/ibatorPlugin.html) page for more information.
* Added "delimitAllColumns" attribute to table configurations. This supports databases like PosgreSQL that are case sensitive for identifiers. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) page for more information.
* Added a page explaining how to deal with case sensitivity in PostgreSQL. See the [PostgreSQL](http://mbg.cndocs.tk/usage/postgresql.html) page for more information.
* IBATIS-586 - Added the ability to specify nested property elements on columnOverrides. See the [<columnOverride>](http://mbg.cndocs.tk/configreference/columnOverride.html) page for more information. Thanks to Dan Turkenkopf for the idea and a nice initial patch.
* The IntrospectedColumn class now contains any column remarks returned during database introspection. This may be useful for some CommentGenerators.
* IBATIS-592 - Added attributes to IntrospectedTable that contain the calculated SqlMap namespace, and the calculated runtime table names. These can now be overridden in plugins.
* Fixed addCriterionfor JDBC\* methods so that they all do a null check.
* Fixed IbatorRunner so that configuration errors are shown (thanks to Karel Rank)
* IBATIS-605 - Added Informix Dialect for [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html)
* Addedd support for "distinct" on select by example methods
* Added new "or" method to example classes
* Added new "useCompoundPropertyNames" property on <table>
* Enabled a far simpler method for extending the example classes
* EqualsHashCodePlugin now generates far superior methods

**Version 1.2.1**

**Bugs 修复**

* Fixed the IbatorObjectFactory so it will find internal classes from the context classloader.
* Fixed IBATIS-565 - ill formed comment in the SqlMapConfigPlugin

**改进**

* Modified plugin methods for model fields, getters, and setters so that the plugin will know which type of class (Primary Key, Base Record, or Record with BLOBs) is being generated.
* Added methods to IntrospectedTable to get/set attributes. This allows plugin classes to maintain table based state between plugin calls.
* Added initialized method to the plugin API. This allows plugins to alter some of the fundamental code generation items (like the name of a generated class, for example).
* Added an example plugin to show usage of the initialized method.

**Version 1.2.0**

**Announcements**

* With version 1.2, Abator is renamed to Apache iBATIS Ibator. Several changes have been made to the XML configuration as well as the Java API. See the [Migrating from Abator](http://mbg.cndocs.tk/migratingFromAbator.html) page for detailed information about changes needed to existing Abator configuration files.

**Bugs 修复**

* Fixed the JavaTypeResolver so that columns with unsupported data types may be overridden by configuration.
* Fixed IBATIS-523 - a bug in the pre-release version of the EqualsHashCodePlugin
* Fixed IBATIS-542 - upgrade the build to Ant version 1.7.1

**改进**

* Ibator now includes a plugin mechanism. This mechanism can be used to add to or modify the code generated by Ibator. If you have previously extended one of Abator's code generators to change their behavior, we strongly suggest that you move to a plugin. See the [<ibatorPlugin>](http://mbg.cndocs.tk/configreference/ibatorPlugin.html) page for detailed information.
* Ibator ships with the following plugins:
  + A plugin that will generate an SQL Map Config File (org.apache.ibatis.ibator.plugins.SqlMapConfigPlugin)
  + A plugin that will make generated model classes Serializable (org.apache.ibatis.ibator.plugins.SerializablePlugin)
  + A plugin that will add equals and hashCode methods to generated model classes (org.apache.ibatis.ibator.plugins.EqualsHashCodePlugin)
* Added support for "runtimeCatalog" and "runtimeSchema" properties to the [<table>](http://mbg.cndocs.tk/configreference/table.html) configuration element. Thanks to Dan Turkenkopf for the idea and the patch!
* New generated method - insertSelective. This method will allow you to use column defaults on a table definition on insert
* Added the ability to specify a that DAO implementation classes be in a separate package from the DAO interface classes.

**Changes from Abator**

There are several breaking changes between Ibator and Abator. This list details the changes, and has methods of resolving the differences.

* JSE 5.0 or higher is required for Ibator
* Ibator does not contain the "legacy" code generators from Abator. You must choose "Ibatis2Java2" or "Ibatis2Java5" as a target runtime - and code generated from Ibator is compatible with iBATIS version 2.2.0 or higher only. If you are using an earlier version of iBATIS - upgrade! If you are not able to upgrade, then you must continue to use Abator.
* The classloading strategy in Ibator is changed from Abator. In all cases, we now recommend specifying the classpath external to Ibator and we further recommend that you do not use the <classPathEntry> element. You may specify classpath entries if you feel you must, but those entries will only be used when loading JDBC drivers of Java model root classes. If you write a custom extension to Ibator, or a plugin, you must specify that classpath entry external to Ibator.
* The API for extending Ibator is significantly changed from Abator. In most cases, implementations of the old Abator interfaces should be converted to Ibator plugins. See the [Extending Ibator](http://mbg.cndocs.tk/reference/extending.html) for more information.
* The afterXXXGenerationHook methods have been removed from all Ibator supplied implementations of the core interfaces. If you were extending an Ibator supplied implementation to make use of these methods, then you *must* migrate your code to an Ibator plugin.
* The build has been significantly modified and now includes an Emma based code coverage report.
* Changes to the XML configuration file are required. See the [Migrating from Abator](http://mbg.cndocs.tk/migratingFromAbator.html) page for detailed information

**Version 1.1.0**

**Announcements**

* The next release of Abator will require JRE 5.0 or higher.
* Java2 is now the default generator set. This will cause different code to be generated if you have not specified a generator set previously. To remedy this, set the generator set to "Legacy".

**New Generated Methods**

Abator will generate these new methods:

**countByExample**

This method will return an integer representing the number of rows in a table that match the given criteria.

**updateByExample**

This method will update all rows in a table that match a given criteria. This method is available in the Java2 and Java5 generator sets only. There is also a "selective" version of the method that only updates certain columns of a table (the selective version of this method is probably the more useful version to use in most situations).

**Bugs 修复**

* Fixed bug for corner case where a criteria class is created, but no criteria are set.
* Fixed a bug that caused the JavaModelGenerator's "trimStrings" property to fail
* Fixed the XML file merger so that internal entities are preserved
* Fixed the XML configuration parser so that external entities are handled properly
* Fixed bug - incorrect datatype mapping for JDBC BIT datatype
* Fixed bug where Abator generated incorrect properties for certain database columns (for example, if the column name is I\_NAME)

**Miscellaneous Changes**

* Added the ability to specify properties to ignore qualifiers and change runtime table names in the generated SQL for a table. Primary use cases for this support include:
  + Generating objects for a table that has a public synonym or alias
  + Generating objects for a table that exists in many schemas, and the schema will be selected at runtime

See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information, or the [Oracle](http://mbg.cndocs.tk/usage/oracle.html) reference page for an example.

* Added support for delimiting SQL identifiers for the use cases where identifiers contain spaces or SQL reserved words. See the [<table>](http://mbg.cndocs.tk/configreference/table.html), [<abatorContext>](http://mbg.cndocs.tk/configreference/abatorContext.html), and [<columnOverride>](http://mbg.cndocs.tk/configreference/columnOverride.html) reference pages for more information.
* Added SYBASE dialect for generated keys. See the [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html) reference page for more information.
* Added DB2\_MF (DB2 on Main Frames) dialect for generated keys. See the [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html) reference page for more information.
* Abator will now automatically escape identifiers that contain the $ or # characters as these characters have special meaning in iBATIS configuration files.
* Added a clear method to the generated example classes (in the Java2 and Java5 generator sets only). This allows reuse of these classes.
* Added the ability to specify that result maps should use column indexes rather than column names in the result mappings. Primary use cases for this support include:
  + When tables have columns whose name is only differentiated by case (e.g. "first name" and "First Name")
  + When you want to make the selects as fast as possible (there is a slight performance benefit when using column indexes)

See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information.

* Made the generated Example and Criteria classes extendable. Added some documentation about how to extend these classes. See the [Extending the Example Classes](http://mbg.cndocs.tk/generatedobjects/extendingExampleClass.html) reference page for more information.
* Made the legacy DAOs extendable.
* Added the ability to provide a renaming rule for columns. This is for the use case where columns have a common prefix that should be removed before calculating the property name. See the [<columnRenamingRule>](http://mbg.cndocs.tk/configreference/columnRenamingRule.html) reference page for more information.
* Added support for persisting a configuration to XML - this to enable a graphical editor in the future.
* Add afterXXXGenerationHook() methods in all generators to enable adding extra Java code or XML elements to any generated object. This will make it easier to create customized generators.
* API change to allow generating with selected contexts rather than the entire config file.
* API change to allow generating with selected tables rather than the entire config file.
* Exposed new support for selecting tables and/or contexts to the command line and the Ant task - this has added an advanced syntax to the command line for Abator. See the [Running Abator](http://mbg.cndocs.tk/running.html) reference page for more information.
* rootClass and rootInterface may now be specified for each table. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information.
* If a rootClass is specified for any table, Abator will now check in the rootClass to see if a generated property already exists in the root class. If it does, Abator will not generate the property. The <javaModelGenerator> element now accepts a property to specify the classpath of the rootClass. See the [<javaModelGenerator>](http://mbg.cndocs.tk/configreference/javaModelGenerator.html) reference page for more information. Thanks to Ashok Madhavan for the beginnings of this code.
* Allowed specifying a type (pre or post) for the generated key element. See the [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html) reference page for more information.
* Added a comment generator interface to enable generation of custom comments. See the [<commentGenerator>](http://mbg.cndocs.tk/configreference/commentGenerator.html) reference page for more information.

**Version 1.0.0**

**Generator Sets**

A generator set is a set of code generators (SQL Map Generator, Java Model Generator, DAO Generator, and Java Type Resolver). Abator now ships three different generator sets. The generated code from these three generator sets is slightly different, and the use of the generated objects is slightly different too. The concepts are exactly the same. With the newer generator sets, the "by example" methods have been vastly improved. It is now possible to generate **virtually any** WHERE clause (including IN and BETWEEN predicates). Lastly, the new generator sets generate much more concise code - the DAOs and SQL Maps are of normal size, and there are no extraneous methods. The example class in the new generator sets encapsulates all the function needed to generate dynamic queries.

The three generator sets shipped with Abator are as follows:

Legacy

This generator set generates code that is the same as previous versions of Abator. There are some limitations with this generator set and we strongly recommend that you choose one of the other sets. However, this set remains the default for now. **This generator set will likely be removed in a future version of Abator.**

Java2

This generator set generates code that is compatible with iBATIS versions 2.2.0 and higher. With this generator set the "by example" methods are much more powerful than in the legacy set. It is now possible to generate virtually unlimited SQL WHERE clauses with Abator generated code (including "IN" and "BETWEEN" clauses). **This generator set will likely become the default set in a future version of Abator.**

Java5

This generator set has the same capabilities as the Java2 generator set with the added feature of generating code that is JSE 5.0 compliant using parameterized types and annotations.

**重要:** code generated with the Java2 or Java5 generator sets is not 100% compatible with code generated with the Legacy set - especially in the use of the "by example" methods. Also note that the "by example" methods in these generator sets rely on iBATIS dynamic SQL support that is missing in iBATIS versions prior to version 2.2.0.

A generator set is selected with the generatorSet attribute of the <abatorContext> element. See the [<abatorContext>](http://mbg.cndocs.tk/configreference/abatorContext.html) reference page for more information.

Use of the example classes is different with the different generator sets. See the [Example Class Usage](http://mbg.cndocs.tk/generatedobjects/exampleClassUsage.html) page for more information.

**Model Types**

A model type is used to give you more control over the types of domain objects generated by Abator. Abator now supports three different types of domain models as follows:

conditional

This model is similar to the hierarchical model except that a separate class will not be generated if that separate class would only contain one field. So if a table has only one primary key field, that field will be merged into the base record class. This model type is the default. **Note that this model type may generate classes that are not 100% with classes generated in previous versions of Abator.**

flat

This model generates only one domain class for any table. The class will hold all fields in the table.

hierarchical

This model is the same as the model shipped with the initial versions of Abator. This model will generate a primary key class if the table has a primary key, another class that holds any BLOB columns in the table, and another class that holds the remaining fields. There is an appropriate inheritance relationship between the classes.

Model types can be specified as a default for an entire context, and you may override the default for each table in a context. See the [<abatorContext>](http://mbg.cndocs.tk/configreference/abatorContext.html) reference page for more information about setting the context default.. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information about setting a model type for specific tables.

**重要:** the default value is conditional - this is a non-backward compatible change from previous versions of Abator.

**updateByPrimaryKeySelective**

This is a new mapped SQL statement, and new DAO method, that will only update columns whose corresponding properties in the parameter class are non-null. This can be used to update certain columns in a record without needing to update the entire record.

**重要:** any column that is mapped to a primitive type will always be updated.

**Miscellaneous Changes**

* Added the ability to specify a table alias. This aids in reuse of generated SQL map elements. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information.
* Fixed the XML file merger so that extraneous blank lines at the end of the file are removed.
* Added the ability to specify a type handler for table columns. See the [<columnOverride>](http://mbg.cndocs.tk/configreference/columnOverride.html) reference page for more information.
* Added the ability to specify the visibility of the DAO "by example" methods. This allows you to make the methods private for internal use only. See the [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) reference page for more information.
* Added the ability to override the naming convention for DAO method names. See the [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) reference page for more information.
* Added the ability to specify wildcards for schema or tableName in a table configuration. This will allow generation of many tables with a simple XML configuration. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information.
* Added the ability to escape wildcard characters in schema or table names if required by the driver. See the [<table>](http://mbg.cndocs.tk/configreference/table.html) reference page for more information.
* Added the ability to suppress JSE 5.0 type warning messages for non-parameterized types if you are using the Legacy or Java2 generator sets in a JSE 5.0 environment. See the [<abatorContext>](http://mbg.cndocs.tk/configreference/abatorContext.html) reference page for more information.
* Added the ability to specify an external properties file for passing parameters into an Abator configuration file (like the iBATIS properties file). See the [<properties>](http://mbg.cndocs.tk/configreference/properties.html) reference page for more information.
* The Ant task now supports a "verbose" attribute. See the [Running Abator](http://mbg.cndocs.tk/running.html) page for more information.
* The Ant task now supports a nested property set for passing values into an Abator configuration file. See the [Running Abator](http://mbg.cndocs.tk/running.html) page for more information.

**MyBatis Generator 快速入门指南**

若要启动并快速运行 MyBatis Generator（MBG），请按以下步骤进行:

1. 创建并填写适当的配置文件。 至少，您必须指定:
   1. <jdbcConnection> 元素定义如何连接目标数据库
   2. <javaModelGenerator> 元素来指定生成 Java 模型对象所属的包
   3. <sqlMapGenerator> 元素来指定生成 SQL 映射文件所属的包和的目标项目
   4. (可选的) <javaClientGenerator> 元素来指定目标包和目标项目生成的客户端接口和类 （如果您不想生成 Java 客户端代码您可以省略< javaClientGenerator > 元素）

查看 [XML 配置文件参考](http://mbg.cndocs.tk/configreference/xmlconfig.html) 一个配置文件的例子.

1. 将文件保存在一些方便的位置(例如 \temp\generatorConfig.xml)
2. 按下面的方式从命令行运行 MBG:
3. java -jar mybatis-generator-core-x.x.x.jar -configfile \temp\generatorConfig.xml -overwrite

这会告诉 MBG 使用配置文件去运行。 MBG还会覆盖已经存在的同名Java文件。 如果您想保留已经存在的Java文件，您可以忽略 -overwrite 参数。 如果存在冲突, MBG 会用一个唯一的名字保存新生成的文件（例如：MyClass.java.1)。

1. MBG运行后, 您将需要创建或修改的标准 MyBatis 或 iBATIS 配置文件来使用新生成的代码。查看 [运行 MyBatis Generator 后的任务](http://mbg.cndocs.tk/afterRunning.html) 页面获取更多信息。

**重要:**为iBATIS2生成的代码需要在您的 iBATIS 配置中启用了该命名空间。 查看 [运行 MyBatis Generator 后的任务](http://mbg.cndocs.tk/afterRunning.html) 页面获取更多信息。

**运行 MyBatis Generator**

MyBatis Generator (MBG) 可以通过以下方式运行:

* 从 [命令提示符](http://mbg.cndocs.tk/running/runningFromCmdLine.html) 使用 XML 配置文件
* 作为 [Ant 任务](http://mbg.cndocs.tk/running/runningWithAnt.html) 使用 XML 配置文件
* 作为 [Maven Plugin](http://mbg.cndocs.tk/running/runningWithMaven.html)
* 从另一个 [Java 程序](http://mbg.cndocs.tk/running/runningWithJava.html) 使用 XML 配置文件

每种方法的详细信息都在链接的页面上。

**注意:**还有一个可以良好的集成到Eclipse的MBG插件可以提供额外的功能。 Eclipse 启用 Ant 任务，并支持 Java 文件的自动合并。安装Eclipse插件请参考 [MyBatis](http://mybatis.org)站点页面的信息。

**重要:** 当运行在像Eclipse这样的IDE环境的外部运行时, MBG解释targetProject 和 targetPackage的所有XML属性如下:

* targetProject 被假定为一个已存在的目录结构。 如果目录结构不存在MBG将会失败。有一个例外 - 当MBG通过Maven插件运行时。 请从Maven插件页面参阅targetProject在Maven中的详细解释。
* targetPackage 将会转换为 targetProject 适当的子目录结构。 如果有必要，MBG会创建这些子目录。

# 从命令行运行 MyBatis Generator

MyBatis Generator (MBG) 可以直接从命令行运行。JAR清单包括默认类的名称 (org.mybatis.generator.api.ShellRunner) 或者您可以自己指定它。 ShellRunner 类接受以下的几种详细参数:

|  |  |
| --- | --- |
| **参数** | **值** |
| -configfile *file\_name*  (必须的) | 指定配置文件的名称。 |
| -overwrite (可选的) | 如果指定了该参数，如果生成的java文件存在已经同名的文件，新生成的文件会覆盖原有的文件。 如果没有指定该参数，如果存在同名的文件，MBG会给新生成的代码文件生成一个唯一的名字(例如： MyClass.java.1, MyClass.java.2 等等)。 **重要: 生成器一定会自动合并或覆盖已经生成的XML文件。** |
| -verbose (可选的) | 如果指定该参数，执行过程会输出到控制台。 |
| -forceJavaLogging (可选的) | 如果指定该参数，MBG将会使用JAVA日志记录而不会使用Log4J,即使Log4J在运行时的类路径中。 |
| -contextids *context1,context2,...* (可选的) | 如果指定了该参数，逗号隔开的这些context会被执行。 这些指定的context必须和配置文件中 <context> 元素的 id 属性一致。 只有指定的这些contextid会被激活执行。如果没有指定该参数，所有的context都会被激活执行。 |
| -tables *table1, table2,...* (可选的) | 如果指定了该参数，逗号隔开的这个表会被运行， 这些表名必须和 <table> 配置中的表面完全一致。只有指定的这些表会被执行。 如果没有指定该参数，所有的表都会被执行。 按如下方式指定表明:   table schema.table catalog..table 等等。 |

从命令行运行MGB时您必须指定 XML 配置文件。如果文件的名字是"generatorConfig.xml",可以用下面任意的命令执行:

java -jar mybatis-generator-core-x.x.x.jar -configfile generatorConfig.xml

java -jar mybatis-generator-core-x.x.x.jar -configfile generatorConfig.xml -overwrite

java -cp mybatis-generator-core-x.x.x.jar org.mybatis.generator.api.ShellRunner -configfile generatorConfig.xml

java -cp mybatis-generator-core-x.x.x.jar org.mybatis.generator.api.ShellRunner -configfile generatorConfig.xml -overwrite

**使用Ant运行 MyBatis Generator**

MyBatis Generator (MBG) 包含一个简单的 Ant 任务。这个任务必须定义在您的 build.xml 文件中。任务可以几个参数。这里是一个 build.xml 文件的例子:

<project default="genfiles" basedir=".">

<property name="generated.source.dir" value="${basedir}" />

<target name="genfiles" description="Generate the files">

<taskdef name="mbgenerator"

classname="org.mybatis.generator.ant.GeneratorAntTask"

classpath="mybatis-generator-core-x.x.x.jar" />

<mbgenerator overwrite="true" configfile="generatorConfig.xml" verbose="false" >

<propertyset>

<propertyref name="generated.source.dir"/>

</propertyset>

</mbgenerator>

</target>

</project>

MyBatis Generator 任务属性如下:

|  |  |
| --- | --- |
| **属性** | **值** |
| configfile (必须的) | 指定配置文件的名称。 |
| overwrite (可选的) | 如果指定了该参数，如果生成的java文件存在已经同名的文件，新生成的文件会覆盖原有的文件。 如果没有指定该参数，如果存在同名的文件，MBG会给新生成的代码文件生成一个唯一的名字(例如： MyClass.java.1, MyClass.java.2 等等)。 **重要: 生成器一定会自动合并或覆盖已经生成的XML文件。** |
| contextids (可选的) | 如果指定了该参数，逗号隔开的这些context会被执行。 这些指定的context必须和配置文件中 <context> 元素的 id 属性一致。 只有指定的这些contextid会被激活执行。如果没有指定该参数，所有的context都会被激活执行。 |
| tables (可选的) | 如果指定了该参数，逗号隔开的这个表会被运行， 这些表名必须和 <table> 配置中的表面完全一致。只有指定的这些表会被执行。 如果没有指定该参数，所有的表都会被执行。 按如下方式指定表明:   table schema.table catalog..table 等等。 |
| verbose (可选的) | 如果设置为 "true", "yes"等等,MBG会将操作信息输出到 ant 控制台 (如果 Ant 在verbose模式下运行)。默认值是 "false"。 |

注意:

* <taskdef>标记中的类路径用于告诉 Ant MBG JAR 文件在哪里。 这是可选的，除非您将 MBG 添加到 Ant 类路径中 Ant 手册所描述的其他地方。
* 任务的名称可以使任何您想要的, "mbgenerator" 只是一个简单的例子。
* 该任务支持可选嵌套的 < propertyset > 元素，这是标准的 Ant 属性设置的类型。 这可以用于将参数传递到配置文件。例如，上面提到 generated.source.dir这个属性可以在配置文件中使用${generated.source.dir}进行使用。
* 如果没有指定配置文件中的一个属性，这个属性将会原样输出。

**通过Maven运行 MyBatis Generator**

MyBatis Generator (MBG) 包含了一个可以集成到Maven构建的Maven插件，按照Maven的配置惯例, 将MBG集成到Maven很容易. 最简配置如下:

<project ...>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.0</version>

</plugin>

...

</plugins>

...

</build>

...

</project>

当然，事情永远不会那么容易！

**Maven Goal and Execution（Maven 目标和执行）**

The MBG Maven plugin 包含一个目标:

* mybatis-generator:generate

这个目标不会被Maven自动执行，他可以通过以下两种方式执行。

可以在命令行通过以下命令执行:

* mvn mybatis-generator:generate

您可以通过标准的Maven命令属性传递参数， 例如:

* mvn -Dmybatis.generator.overwrite=true mybatis-generator:generate

这条命令会使MBG覆盖重名的文件

在一个连续的编译环境中，您可能想让MGB作为Maven构建的一部分自动执行。 这可以通过配置自动执行的目标来实现，这种情况的例子如下:

<project ...>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.0</version>

<executions>

<execution>

<id>Generate MyBatis Artifacts</id>

<goals>

<goal>generate</goal>

</goals>

</execution>

</executions>

</plugin>

...

</plugins>

...

</build>

...

</project>

MBG插件将会绑定到Maven构建的 generate-sources 阶段。 所以他会在编译步骤之前执行。 此外注意MBG目标将绑定生成Java和XML资源文件的建立，他们都将包括在生成的JAR包内。

**MyBatis Generator 配置属性**

所有配置在POM中的属性都可以传递到配置文件，并且可以用通常的方式使用。例如:

<project ...>

...

<properties>

<dao.target.dir>src/main/java</dao.target.dir>

</properties>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.0</version>

<executions>

<execution>

<id>Generate MyBatis Artifacts</id>

<goals>

<goal>generate</goal>

</goals>

</execution>

</executions>

</plugin>

...

</plugins>

...

</build>

...

</project>

在这种情况下，属性可以在配置文件中被访问的语法是 ${dao.target.dir}.

**参考参数**

所有的参数都是可选的，大部分都适合的默认值。

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **表达式** | **类型** | **注释** |
| configurationFile | ${mybatis.generator.configurationFile} | java.io.File | 指定配置文件的名称。  默认值:  ${basedir}/src/main/resources/generatorConfig.xml |
| contexts | ${mybatis.generator.contexts} | java.lang.String | 如果指定了该参数，逗号隔开的这些context会被执行。 这些指定的context必须和配置文件中 <context> 元素的 id 属性一致。 只有指定的这些contextid会被激活执行。如果没有指定该参数，所有的context都会被激活执行。 |
| jdbcDriver | ${mybatis.generator.jdbcDriver} | java.lang.String | 如果您指定了 sqlScript 参数, 当连接数据库时这里的值是JDBC驱动类的权限定名称。 |
| jdbcPassword | ${mybatis.generator.jdbcPassword} | java.lang.String | 如果您指定了 sqlScript 参数, 这是连接数据库的密码。 |
| jdbcURL | ${mybatis.generator.jdbcURL} | java.lang.String | 如果您指定了 sqlScript 参数, 这是连接数据库的JDBC URL |
| jdbcUserId | ${mybatis.generator.jdbcUserId} | java.lang.String | 如果您指定了 sqlScript 参数, 这里是连接数据库的用户id |
| outputDirectory | ${mybatis.generator.outputDirectory} | java.io.File | 将放置 MBG 所生成文件的目录。 这个目录是用于当 targetProject 在配置文件中设置特殊值的"MAVEN"时使用(大小写敏感)。  默认值:  ${project.build.directory}/generated-sources/mybatis-generator |
| overwrite | ${mybatis.generator.overwrite} | boolean | 如果指定了该参数，如果生成的java文件存在已经同名的文件，新生成的文件会覆盖原有的文件。 如果没有指定该参数，如果存在同名的文件，MBG会给新生成的代码文件生成一个唯一的名字(例如： MyClass.java.1, MyClass.java.2 等等)。 **重要: 生成器一定会自动合并或覆盖已经生成的XML文件。**  默认值:  false |
| sqlScript | ${mybatis.generator.sqlScript} | java.lang.String | 要在生成代码之前运行的 SQL 脚本文件的位置。 如果空，不会执行任何脚本。 如果不是空，jdbcDriver, jdbcURL 参数必须提供。 另外如果连接数据库需要认证也需要提供 jdbcUserId 和 jdbcPassword 参数。  值可以使一个文件系统的绝对路径或者是一个使用"classpath:"开头放在构建的类路径下的路径。 |
| tableNames | ${mybatis.generator.tableNames} | java.lang.String | 如果指定了该参数，逗号隔开的这个表会被运行， 这些表名必须和 <table> 配置中的表面完全一致。只有指定的这些表会被执行。 如果没有指定该参数，所有的表都会被执行。 按如下方式指定表明:   table schema.table catalog..table 等等。 |
| verbose | ${mybatis.generator.verbose} | boolean | 如果指定该参数，执行过程会输出到控制台。 |

**targetProject 解释**

与Maven运行时 生成器 配置的 targetProject 属性有不同的解释。 如果指定值为"MAVEN" (大小写敏感), targetProject 将被设置为插件的输出目录，而且如果不存在这个目录，将会创建这个目录。 如果没有设置为 "MAVEN", 那么targetProject 将会被 MGB 当成普通的 - 它必须是一个已经存在的目录。

**使用Java运行 MyBatis Generator**

MyBatis Generator (MBG) 可以直接使用Java调用。 对于配置，您可以使用XML配置文件，或者完全使用Java进行配置。

**使用XML配置文件从Java运行MBG**

下面的代码例子展示了如何通过XML配置文件从Java运行MBG。 他不显示异常处理，但是编译错误是很明显的 :)

List<String> warnings = new ArrayList<String>();

boolean overwrite = true;

File configFile = new File("generatorConfig.xml");

ConfigurationParser cp = new ConfigurationParser(warnings);

Configuration config = cp.parseConfiguration(configFile);

DefaultShellCallback callback = new DefaultShellCallback(overwrite);

MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);

myBatisGenerator.generate(null);

注意:

* 配置文件属性可以通过 ConfigurationParser 的构造函数的参数传递给解析器。如果没有显式传递，配置文件的属性将会从JVM的系统属性搜索。 例如，属性 generated.source.dir 可以在配置文件中通过 ${generated.source.dir} 被访问。
* 如果没有指定配置文件中的一个属性，这个属性将会原样输出。

**通过基于Java的配置运行MGB**

下面的代码例子展示了如何通过基于Java的配置运行MGB。 他不显示异常处理，但是编译错误是很明显的 :)

List<String> warnings = new ArrayList<String>();

boolean overwrite = true;

Configuration config = new Configuration();

// ... fill out the config object as appropriate...

DefaultShellCallback callback = new DefaultShellCallback(overwrite);

MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);

myBatisGenerator.generate(null);

**运行 MyBatis Generator 后的任务**

在您运行 MyBatis Generator (MBG)后, 您需要创建或者修改其他的 MyBatis 或 iBATIS 配置文件。 主要的任务如下:

* 对 MyBatis 3.x:
  + 创建或修改 MapperConfig.xml 文件
* 对 iBATIS 2.x:
  + 创建或修改 SqlMapConfig.xml 文件
  + 创建或修改 dao.xml 文件 (只有当您使用iBATIS DAO 框架时)

下面将详细地介绍每个任务。

**更新 MapperConfig.xml 文件 (MyBatis 3.x)**

MyBatis 3.x 使用了一个 XML 文件, 通常的名字是 MapperConfig.xml, 声明数据库连接信息, 事务管理方案, 和 将会用在MyBatis会话中的 XML mapper 文件。 MBG 不会为您创建这个文件，因为它对您执行环境一无所知。然而，一些在此文件中的项目直接涉及 MBG 生成项目。 请参阅标准 MyBatis 数据映射开发指南有关不同的配置选项的详细信息。

MBG 具体需要在配置文件中的有如下:

* 必须列出MBG 生成的 XML 映射文件

例如,假设MBG已经生成一个名为MyTableMapper.xml的XML映射文件, 并且该文件已经在您的项目的test.xml包中。 MapperConfig.xml 文件应该有这些条目:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 安装适合于您的环境的事务管理器和数据源 -->

<environments default"...">

<environment id"...">

<transactionManager type="...">

</transactionManager>

<dataSource type="...">

</dataSource>

</environment>

</environments>

<mappers>

<!-- XML mapper 文件应该在这里列出来 -->

<mapper resource="test/xml/MyTable\_SqlMap.xml" />

</mappers>

</configuration>

如果有多个XML mapper文件(这很常见)， 然后您可以通过反复的使用<mapper> 按任意顺序在<mappers>元素中列出这些文件。

**生成 MapperConfig.xml** 您可能会要MBG使用MapperConfigPlugin生成一个Mapper配置文件的骨架。 查看[<插件>](http://mbg.cndocs.tk/configreference/plugin.html)页面获取详细信息。

**Updating the SqlMapConfig.xml File (iBATIS 2.x)**

iBATIS 2 uses an XML file, commonly named SqlMapConfig.xml, to specify information for a database connection, a transaction management scheme, and SQL map XML files that will be used in an iBATIS session. MBG cannot create this file for you because MBG knows nothing about your execution environment. However, some of the items in this file relate directly to MBG generated items. Please refer to the standard iBATIS data mapper developer guide for details about the different configuration options.

MBG specific needs in the configuration file are as follows:

* Statement namespaces must be enabled
* MBG generated SQL Map XML files must be listed

For example, suppose that MBG has generated an SQL Map XML file called MyTable\_SqlMap.xml, and that the file has been placed in the test.xml package of your project. The SqlMapConfig.xml file should have these entries:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sqlMapConfig

PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

<!-- Statement namespaces are required for MBG -->

<settings useStatementNamespaces="true" />

<!-- Setup the transaction manager and data source that are

appropriate for your environment

-->

<transactionManager type="...">

<dataSource type="...">

</dataSource>

</transactionManager>

<!-- SQL Map XML files should be listed here -->

<sqlMap resource="test/xml/MyTable\_SqlMap.xml" />

</sqlMapConfig>

If there is more than one SQL Map XML file (as is quite common), then the files can be listed in any order with repeated <sqlMap> elements after the <transactionManager> element.

**Version 1.2 New Enhancement** With MBG version 1.2 and later, You may ask MBG to generate a skeleton SQL Map Configuration file with the SqlMapConfigPlugin. See the [<plugin>](http://mbg.cndocs.tk/configreference/plugin.html) page for more information.

**Updating the dao.xml File (iBATIS 2.x)**

**重要提示:** this step is only required if you generated DAOs for the deprecated iBATIS DAO framework (we suggest using Spring instead).

The iBATIS DAO framework is configured by an xml file commonly called dao.xml. The iBATIS DAO framework uses this file to control the database connection information for DAOs, and also to list the DAO implementation classes and DAO interfaces. In this file you should specify the path to your SqlMapConfig.xml file, and all the MBG generated DAO interfaces and implementation classes.

For example, suppose that MBG has generated a DAO interface called MyTableDAO and a implementation class called MyTableDAOImpl, and that the files have been placed in the test.dao package of your project. The dao.xml file should have these entries:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE daoConfig

PUBLIC "-//ibatis.apache.org//DTD DAO Configuration 2.0//EN"

"http://ibatis.apache.org/dtd/dao-2.dtd">

<daoConfig>

<context>

<transactionManager type="SQLMAP">

<property name="SqlMapConfigResource"

value="test/SqlMapConfig.xml"/>

</transactionManager>

<!-- DAO interfaces and implementations should be listed here -->

<dao interface="test.dao.MyTableDAO"

implementation="test.dao.MyTableDAOImpl" />

</context>

</daoConfig>

**MyBatis Generator XML 配置参考**

最常见的用例中，代码生成器(MBG)是由一个XML配置文件驱动。配置文件告诉MBG:

* 如何连接到数据库
* 生成什么对象，以及如何生成它们
* 那些表生成对象

下面是一个MBG配置文件的例子。查阅每个元素单独的页面查看更多有关元素的更多信息和属性值。

<?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>

<classPathEntry location="/Program Files/IBM/SQLLIB/java/db2java.zip" />

<context id="DB2Tables" targetRuntime="MyBatis3">

<jdbcConnection driverClass="COM.ibm.db2.jdbc.app.DB2Driver"

connectionURL="jdbc:db2:TEST"

userId="db2admin"

password="db2admin">

</jdbcConnection>

<javaTypeResolver >

<property name="forceBigDecimals" value="false" />

</javaTypeResolver>

<javaModelGenerator targetPackage="test.model" targetProject="\MBGTestProject\src">

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

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

</javaModelGenerator>

<sqlMapGenerator targetPackage="test.xml" targetProject="\MBGTestProject\src">

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

</sqlMapGenerator>

<javaClientGenerator type="XMLMAPPER" targetPackage="test.dao" targetProject="\MBGTestProject\src">

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

</javaClientGenerator>

<table schema="DB2ADMIN" tableName="ALLTYPES" domainObjectName="Customer" >

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

<generatedKey column="ID" sqlStatement="DB2" identity="true" />

<columnOverride column="DATE\_FIELD" property="startDate" />

<ignoreColumn column="FRED" />

<columnOverride column="LONG\_VARCHAR\_FIELD" jdbcType="VARCHAR" />

</table>

</context>

</generatorConfiguration>

有关此文件的重要说明:

* 该文件指定旧式DB2 CLI驱动程序将用于连接到数据库，并指定在何处可以找到驱动程序。
* Java类型解析器不应该强制型对象字段BigDecimal的使用，这意味着整数类型(短、整型、长等)如果可能的话将被取代。 此功能是为了使数据库DECIMAL和NUMERIC列容易处理。
* Java模型生成器应该使用子包。 这意味着在这种情况下生成的模型对象将被放置在名为 test.model.db2admin 的包中（因为表在 DB2ADMIN schema中）。 如果 enableSubPackages 属性设置为 false， 那么包名将会是 test.model。 Java模型生成器也应该对字符串进行trim操作。 这意味着任何字符串属性的setter方法将调用trim方法 - 如果您的数据库可能会在字符末尾返回空白符，这是非常有用的。
* SQL映射生成器将使用子包。这意味着这种情况下生成的XML文件将被放置在名为 test.xml.db2admin 的包中（因为表在 DB2ADMIN schema中）。 如果 enableSubPackages 属性设置为 false， 那么包名将会是 test.xml。
* DAO生成器将使用子包。这意味着这种情况下生成的DAO类都会被放置在名为 test.dao.db2admin 的包中（因为表在 DB2ADMIN schema中）。 如果 enableSubPackages 属性设置为 false， 那么包名将会是 test.dao。 DAO生成器会生成一些引用Mybatis XML配置的mapper接口。
* The file specifies only one table will be introspected, but many more could be specified. Important notes about the specified table include:
  + The generated objects will be based on the name Customer (CustomerKey, Customer, CustomerMapper, etc.) - rather than on the table name.
  + Actual column names will be used as properties. If this property were set to false (or not specified), then MBG would attempt to camel case the column names. In either case, the name can be overridden by the <columnOverride> element
  + The column has a generated key, it is an identity column, and the database type is DB2. This will cause MBG to generate the proper <selectKey> element in the generated <insert> statement so that the newly generated key can be returned (using DB2 specific SQL).
  + The column DATE\_FIELD will be mapped to a property called startDate. This will override the default property which would be DATE\_FIELD in this case, or dateField if the useActualColumnNames property was set to false.
  + The column FRED will be ignored. No SQL will list the field, and no Java property will be generated.
  + The column LONG\_VARCHAR\_FIELD will be treated as a VARCHAR field, regardless of the actual data type.

**<classPathEntry> 元素**

<classPathEntry> 元素 用于添加运行类路径位置到类路径中的 MyBatis Generator（MBG）。 <classPathEntry> 元素是[<generatorConfiguration>](http://mbg.cndocs.tk/configreference/generatorConfiguration.html) 的子元素. MBG 在这些情况下从这些路径下加载类:

* 当加载 JDBC 驱动内省数据库时
* 当加载根类中的 JavaModelGenerator 检查重写的方法时

这个元素是可选的，而且如果您给MGB安装了类路径，您就不需要这个元素(例如 使用java命令时使用参数-cp)。

**重要说明:** 加载扩展一个 MBG 的类或实现 MBG 的接口之一的类时，不会使用这些位置。 在这种情况下，您必须以相同的方式将 MBG 添加到运行时的类路径下(例如 使用java命令时使用参数-cp)。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| location | 要添加到类路径中的 JAR/ZIP 文件的完整路径名称或要添加到类路径中的目录。 |

**可选属性**

无

**子元素**

无

**例子**

这里指定了DB2 JDBC驱动的路径:

<classPathEntry location="/Program Files/IBM/SQLLIB/java/db2java.zip" />

**<columnOverride> 元素**

MyBatis Generator (MBG) 使用 <columnOverride> 元素从将某些属性默认计算的值更改为内省数据库列得的值。 这个元素是 [<table>](http://mbg.cndocs.tk/configreference/table.html) 元素可选的一个子元素。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| column | 要内省的列名 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| 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"useActualColumnNames"属性(查看[<table>](http://mbg.cndocs.tk/configreference/table.html) 的属性获取更多的信息) 生成"STRT\_DTE" 或 "strtDte"。 |
| javaType | 该列属性的完全限定的 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 分隔的列名称。 如果列的名称中包含空格，MGB会自动的分割列名， 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。  当设置为 "true" 是, column 属性用于重写必须匹配从数据库返回的列名。  分隔符指定在 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素上。  *默认值是false。然而MGB会自动分割包含空格的列名，所以在这种情况下不需要重写。* |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N) 注：在此处定义的任何属性都会被加入相应的内省列的属性集合中。 MBG目前不响应任何属性。 此元素被设置使得对于每一列的特殊值，可以在该插件被编码， 以产生或修改一些独特的特定列中的事件被提供给插件。

**<columnRenamingRule> 元素**

MyBatis Generator (MBG) 使用 <columnRenamingRule> 元素处理在内省表 中相应的属性名之前重命名数据库列。 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。 例子，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。

如果指定该选项，重命名规则会在生成属性名前替换列名。 计算出的属性名可能不同，这取决于 <table> 元素上的 "useActualColumnNames"属性。 下表显示了如果重命名规则用于字段集上时不同的值:

|  |  |  |
| --- | --- | --- |
| **列名** | **属性 useActualColumnNames="true"** | **属性 useActualColumnNames="false"** |
| CUST\_BUSINESS\_NAME | BUSINESS\_NAME | businessName |
| CUST\_STREET\_ADDRESS | STREET\_ADDRESS | streetAddress |
| CUST\_CITY | CITY | city |
| CUST\_STATE | STATE | state |

这个元素是[<table>](http://mbg.cndocs.tk/configreference/table.html) 元素的一个可选的子元素。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| searchString | 定义将被替换的子字符串的正则表达式。 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| 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。 |

**子元素**

无

**<commentGenerator> 元素**

<commentGenerator> 元素用来定义注释生成器的属性。 注释生成器用来给由 MyBatis Generator (MBG) 生成的多种元素（Java自动，Java方法，XML元素等等）生成注释。 默认的注释生成器将 JavaDoc 注释添加到所有生成的Java元素上，从而使Eclipse插件可以使用合并Java的功能。 此外，注释还会添加到每一个生成的XML元素。 注释的目的还在于告诉用户这个元素是生成的，并且有可能重新生成（也就是说 - 他们不应该被修改）。 这个元素是 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素的一个可选的子元素。

默认的实现类是 org.mybatis.generator.internal.DefaultCommentGenerator。 如果您只想修改某些行为，默认的实现类专门为扩展进行设计。

**必选属性**

无

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| type | 使用用户指定提供的注释生成器类型。 这个类必须继承org.mybatis.generator.api.CommentGenerator接口。 而且必须有一个默认的构造函数。 这个属性接收指定特殊的值 DEFAULT，这将会使用默认的实现类（这和不指定该属性的效果一样）。 |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)

**支持的属性**

这个表格列出了注释生成器中所有可以通过[<property>](http://mbg.cndocs.tk/configreference/property.html)子元素进行设置的属性：

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| suppressAllComments | 这个属性用来指定MBG生成的代码中是否包含任何注释。 这个属性有以下可选值：   |  |  | | --- | --- | | **false** | *这是默认值*  当这个属性是false或者没有指定时，所有生成的元素都会包含用来说明这是生成元素的注释。 | | **true** | 当这个属性是true时，不会往生成的元素中添加任何注释。 |   **警告:** 如果您将这个值设为true，那么所有的代码合并都会被禁用。 |
| suppressDate | 这个元素用来指定生成的注释中是否包含生成的日期。 这个属性有以下可选值：   |  |  | | --- | --- | | **false** | *这是默认值*  当这个属性是false或者没有指定时，所有元素生成注释时都会带着生成时间。 | | **true** | 当这个属性是true时，注释中不会添加时间戳。 | |

**示例**

这个元素指定了我们不希望生成的注释中包含时间戳:

<commentGenerator>

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

</commentGenerator>

**<context> 元素**

<context> 元素用于指定生成一组对象的环境。 子元素用于指定要连接到的数据库、 要生成对象的类型和要内省的表。 多个 <context> 元素可以在 [<generatorConfiguration>](http://mbg.cndocs.tk/configreference/generatorConfiguration.html) 元素中列出来，这样可以在同一个MyBatis Generator (MBG)从不同的数据库或者使用不同的生成生成器参数生成对象。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| id | context唯一的标识符。此值将用于某些错误消息。 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| defaultModelType | 这个属性用来设置生成对象类型的默认值。 对象类型定义了MBG如何生成实体类。 对某些对象类型，MBG会给每一个表生成一个单独的实体类。 对另外一些对象类型，MBG会根据表结构生成不同的类。这个属性有以下可选值：   |  |  | | --- | --- | | **conditional** | *这是默认值*  这个模型和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。 | | **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（注：BLOB类 继承 其他字段类 继承 主键类）。 | |
| targetRuntime | 此属性用于指定生成的代码的运行时目标。 该属性支持这些特殊的值：   |  |  | | --- | --- | | **MyBatis3** | *这是默认值*  使用这值的时候，MBG会生成兼容MyBatis 3.0或更高版本， 兼容JSE 5.0或更高版本的对象（例如Java模型类和Mapper接口会使用泛型）。 这些生成对象中的"by example"方法将支持几乎不受限制的动态的where子句。 另外，这些生成器生成的Java对象支持JSE 5.0特性，包含泛型和注解。 | | **MyBatis3Simple** | *这是默认值*  使用这值的时候，和上面的MyBatis3类似，但是不会有"by example"一类的方法，只有少量的动态SQL。 | | **Ibatis2Java2** | 使用这值的时候，MBG会生成兼容iBATIS 2.2.0或更高版本（除了iBATIS 3），还有Java2的所有层次。 这些生成对象中的"by example"方法将支持几乎不受限制的动态的where子句。 这些生成的对象不能100%和原生的Abator或其他的代码生成器兼容。 | | **Ibatis2Java5** | 使用这值的时候，MBG会生成兼容iBATIS 2.2.0或更高版本（除了iBATIS 3）， 兼容JSE 5.0或更高版本的对象（例如Java模型类和Dao类会使用泛型）。 这些生成对象中的"by example"方法将支持几乎不受限制的动态的where子句。 另外，这些生成器生成的Java对象支持JSE 5.0特性，包含泛型和注解。 这些生成的对象不能100%和原生的Abator或其他的代码生成器兼容。 |   如果您想创建一个完全不同的代码生成器， 使用一个继承了org.mybatis.generator.api.IntrospectedTable类的权限定类名替换该值。 通过这个值，您可以创建您自己的代码生成器，然后插入到代码生成器引擎中。 查阅 [扩展 MyBatis Generator](http://mbg.cndocs.tk/reference/extending.html) 页面获取更多信息。 |
| introspectedColumnImpl | 使用这个值去指定一个继承了org.mybatis.generator.api.IntrospectedColumn类的权限定名称。 这可以修改代码生成器计算列信息时候的行为。 查阅 [扩展 MyBatis Generator](http://mbg.cndocs.tk/reference/extending.html) 页面获取更多信息。 |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)
* [<plugin>](http://mbg.cndocs.tk/configreference/plugin.html) (0..N)
* [<commentGenerator>](http://mbg.cndocs.tk/configreference/commentGenerator.html) (0 or 1)
* [<jdbcConnection>](http://mbg.cndocs.tk/configreference/jdbcConnection.html) (1 Required)
* [<javaTypeResolver>](http://mbg.cndocs.tk/configreference/javaTypeResolver.html) (0 or 1)
* [<javaModelGenerator>](http://mbg.cndocs.tk/configreference/javaModelGenerator.html) (1 Required)
* [<sqlMapGenerator>](http://mbg.cndocs.tk/configreference/sqlMapGenerator.html) (0 or 1)
* [<javaClientGenerator>](http://mbg.cndocs.tk/configreference/javaClientGenerator.html) (0 or 1)
* [<table>](http://mbg.cndocs.tk/configreference/table.html) (1..N)

**支持的属性**

下面的表格列出了所有可用的 [<property>](http://mbg.cndocs.tk/configreference/property.html) 子元素:

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| autoDelimitKeywords | 如果是true，那么MGB会分隔SQL关键字，如果他们被用作表中的列名。 MBG维护了许多不同数据库的SQL关键字列表。 然而，列表可能不是非常的全面。 如果一个特殊的关键字没有在MBG的关键字列表中， 您需要通过<columnOverride>强制分隔列。  查看 org.mybatis.generator.internal.db.SqlReservedWords 这个类的源码查看MGB包含的关键字列表。  *默认值是false。* |
| beginningDelimiter | 要用作需要分隔符的 SQL 标识符开头的标识符分隔符的值。 MBG会自动分割包含空格的SQL标识符。 MGB还会自动分隔在 <table> 或 <columnOverride>中配置了具体要求的标识符。  *默认值是双引号 (")。* |
| endingDelimiter | 要使用作为结束的标识符分隔符需要分隔符的 SQL 标识符的值。 MBG会自动分割包含空格的SQL标识符。 MGB还会分隔在 <table> 或 <columnOverride>中配置了具体要求的标识符。  *默认值是双引号 (")。* |
| javaFileEncoding | 通过这个属性设置要使用的Java文件的编码。 新生成的Java文件会用这个编码写入到文件系统。 如果Java文件已经存在并且可以使用这个编码进行读取，就会执行合并操作。 如果没有指定，就会使用平台的默认编码。  查阅 java.nio.charset.Charset 获取可以选择的编码的信息。 |
| javaFormatter | 使用此属性来指定生成的 Java 文件的用户提供 formater 的完整的类名称。 这个类必须继承 org.mybatis.generator.api.JavaFormatter 而且必须有一个默认（不含参数）的构造方法。 每一个context都持有一个单独的javaFormatter的实例。 默认的javaFormatter是 org.mybatis.generator.api.dom.DefaultJavaFormatter。 默认的格式使用内置到Java DOM类的格式。 |
| xmlFormatter | 使用此属性来指定生成的 XML 文件的用户提供 formater 的完整的类名称。 这个类必须继承 org.mybatis.generator.api.XmlFormatter 而且必须有一个默认（不含参数）的构造方法。 每一个context都持有一个单独的xmlFormatter的实例。 默认的xmlFormatter是 org.mybatis.generator.api.dom.DefaultXmlFormatter。 默认的格式使用内置到Java DOM类的格式。 |

# <generatedKey> 元素

<generatedKey> 元素用来指定自动生成主键的属性（identity字段或者sequences序列）。 如果指定了这个元素，MyBatis Generator (MBG)会在生成insert的SQL映射文件中插入一个合适的 <selectKey> 元素。 这个元素是 [<table>](http://mbg.cndocs.tk/configreference/table.html) 元素的一个可选的子元素。

## 必选属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| 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。 或者为您的数据库使用一个合适的语句。 预先定义好的特殊值如下:   |  |  | | --- | --- | | **Cloudscape** | 这将转化为: VALUES IDENTITY\_VAL\_LOCAL() | | **DB2** | 这将转化为: VALUES IDENTITY\_VAL\_LOCAL() | | **DB2\_MF** | 这将转化为:  SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1  为运行在zOS（主框架）或者有些情况是iSeries (AS/400)上的DB2数据库使用该值 | | **Derby** | 这将转化为: VALUES IDENTITY\_VAL\_LOCAL() | | **HSQLDB** | 这将转化为: CALL IDENTITY() | | **Informix** | 这将转化为: select dbinfo('sqlca.sqlerrd1') from systables where tabid=1 | | **MySql** | 这将转化为: SELECT LAST\_INSERT\_ID() | | **SqlServer** | 这将转化为: SELECT SCOPE\_IDENTITY() | | **SYBASE** | 这将转化为: SELECT @@IDENTITY | | **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。 | |

## 可选属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| 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 | 如果指定，则此值将被添加为生成selectKey元素的类型。 此属性的值应该是"pre"或"post"。  **重要:** 如果指定此属性的值，然后生成selectKey元素将永远被放置在insert语句之前。 |

## 子元素

无

**<generatorConfiguration> 元素**

<generatorConfiguration> 元素是MyBatis Generator配置的根元素。这个文件必须包含下面的 DOCTYPE:

<!DOCTYPE generatorConfiguration PUBLIC

"-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

**必选属性**

无

**可选属性**

无

**子元素**

* [<properties>](http://mbg.cndocs.tk/configreference/properties.html) (0 or 1)
* [<classPathEntry>](http://mbg.cndocs.tk/configreference/classPathEntry.html) (0..N)
* [<context>](http://mbg.cndocs.tk/configreference/context.html) (1..N)

# <ignoreColumn> 元素

<ignoreColumn>元素用来告诉 MyBatis Generator (MBG) 忽略一个表中的列。 生成的SQL不会使用该列，而且生成的模型对象中将不包含该列。 这个元素是[<table>](http://mbg.cndocs.tk/configreference/table.html) 元素的一个可选的子元素。

## 必选属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| column | 被忽略的列 |

**<javaClientGenerator> 元素**

<javaClientGenerator> 元素用于定义 Java 客户端代码生成器的属性。 Java客户端生成器生成Java接口和类，它可以简单地使用生成的Java模型和XML映射文件。 在iBATIS2目标环境中，这些对象可以生成DAO接口和实现类形式的代码。 对Mybatis来说，会生成Mapper接口形式的代码。 这个元素是 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素的一个可选的子元素。 如果不指定此元素，然后代码生成器(MBG)将不会生成Java客户端接口和类

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| type | 此属性用于选择一个预定义的Java客户端的生成器，或指定用户提供的Java客户端生成器。 任何用户提供的DAO生成器必须继承 org.mybatis.generator.codegen.AbstractJavaClientGenerator 类, 必须有一个公开默认的构造函数。  该属性接收下列预定义的代码生成器之一:   |  |  | | --- | --- | | 如果<context> targetRuntime 是 **MyBatis3**: | | | **ANNOTATEDMAPPER** | 生成的对象是 MyBatis 3.x 映射器基础结构的 Java接口。 这些接口将会基于注解和 MyBatis 3.x SqlProviders。不会有XML文件生成。（注：就是纯接口使用注解的形式，不会有XML文件）  ANNOTATEDMAPPER 依赖 MyBatis 3.0.4 或更高版本。 | | **MIXEDMAPPER** | 生成的对象是 MyBatis 3.x 映射器基础结构的 Java接口。 这些接口将基于注解和XML的混合形式。 注解将会用在简单注解可以实现的地方。 此客户端不会生成SqlProvider，所有复杂的动态SQL都会生成在XML中。  The MIXEDMAPPER 依赖 MyBatis 3.0.4 或更高版本。 | | **XMLMAPPER** | 生成的对象是 MyBatis 3.x 映射器基础结构的 Java接口。 这些接口将会依赖于生成的XML文件。 | | 如果 <context> targetRuntime 是 **MyBatis3Simple**: | | | **ANNOTATEDMAPPER** | 生成的对象是 MyBatis 3.x 映射器基础结构的 Java接口。 这些接口将会基于注解和 MyBatis 3.x SqlProviders。不会有XML文件生成。（注：就是纯接口使用注解的形式，不会有XML文件）  The ANNOTATEDMAPPER 依赖 MyBatis 3.0.4 或更高版本。 | | **XMLMAPPER** | 生成的对象是 MyBatis 3.x 映射器基础结构的 Java接口。 这些接口将会依赖于生成的XML文件。 | | 如果 <context> targetRuntime 是 **Ibatis2Java2** or **Ibatis2Java5**: | | | **IBATIS** | 生成的对象将符合（不建议使用）iBATIS的DAO框架。 | | **GENERIC-CI** | 生成的对象将只依赖于SqlMapClient。SqlMapClient将会通过构造参数依赖注入提供。 生成的对象将是DAO接口和实现类的形式。 | | **GENERIC-SI** | 生成的对象将只依赖于SqlMapClient。SqlMapClient将会通过setter方法依赖注入提供。 生成的对象将是DAO接口和实现类的形式。 | | **SPRING** | 生成的对象将符合Spring的DAO框架。 | |
| targetPackage | 这个包用于放置生成的接口和实现类。 在默认的生成器中，属性"enableSubPackages"用来控制如何计算实际的包。 如果是true，如果表的catalog和schema存在，就将他们作为子包加起来。 如果"enableSubPackages"是false（默认值），计算的package将是targetPackage属性指定的值。 MBG 将创建所需的生成包的文件夹。  **注:** 实现类的包可以通过指定下面会提到的可选的 implementationPackage 属性。 |
| targetProject | 这用来指定生成接口和类的目标项目。 当在Eclipse环境中运行时，此选项指定保存对象的位置的项目和源文件夹。 在其他环境中，此值应为本地文件系统上的现有目录。 如果它不存在，MBG不会创建它。 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| implementationPackage | 如果指定了该属性，实现类就会生成在这个包中。 在默认的生成器中，属性"enableSubPackages"用来控制如何计算实际的包。 如果是true，如果表的catalog和schema存在，就将他们作为子包加起来。 如果"enableSubPackages"是false（默认值），计算的package将是targetPackage属性指定的值。 MBG 将创建所需的生成包的文件夹。 |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)

**支持的属性**

这个表格列出了所有可以通过[<property>](http://mbg.cndocs.tk/configreference/property.html)子元素进行设置的属性：

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| enableSubPackages | 这个属性用来选择MGB是否根据基于目录和内省表来生成不同的Java包。  例如，假设某个表MYTABLE在MYSCHMA的schema中，也假设targetPackage属性的值设置为"com.mycompany"。 如果此属性为true，为这个表生成的DAO接口和类将被放置在包"com.mycompany.myschema"中。 如果此属性是false，生成的SQL映射将被放在"com.mycompany" schema中。  *默认值是 false* |
| exampleMethodVisibility | 此属性用来设置不同"ByExample"方法的可见性 - selectByExample, deleteByExample等等。 如果没有指定，这些方法将会是公开的的，并将在接口中声明。 此属性使您可以隐藏这些方法如果您只想使用它们来执行其他专门的方法。   |  |  | | --- | --- | | **public** | *这是默认值* 生成实现类中的方法将是公开的，并将该接口中声明的方法。 | | **private** | 生成实现类中的方法将是private的，也不会在接口中声明。 | | **protected** | 生成实现类中的方法将是protected的，也不会在接口中声明。 | | **default** | 生成实现类中的方法将是包内可见的，也不会在接口中声明。 |   **重要提示:** 如果目标运行时是MyBatis3，则将忽略此属性。 |
| methodNameCalculator | 此属性用于选择一个方法名称计算器。 方法名称计算器可以用来为DAO方法提供不同的名称。 您可以选择一个预定义的值。 如果提供的选项都适合您的环境， 也可以指定一个实现了org.mybatis.generator.api.DAOMethodNameCalculator接口的 完全限定名称。   |  |  | | --- | --- | | **default** | *这是默认值* 生成的方法名称会很简单("insert", "updateByPrimaryKey"等等)。 | | **extended** | 生成的方法名称将包括与方法关联的实体对象的名称("insertWidget", "updateWidgetByPrimaryKey"等等)。 |   **重要提示:** this property is ignored if the target runtime is MyBatis3. |
| rootInterface | 此属性用于指定一个所有生成的接口都继承的父接口。 这个值可以通过表配置的 rootInterface 属性覆盖。  **重要:** MBG 不验证接口是否存在或者是否是一个有效的接口。  如果指定，这个属性值应该是一个全限定接口名称(例如 com.github.abel533.mapper.Mapper)。 |
| useLegacyBuilder | 如果为true，那么客户端将使用SqlBuilder为MyBatis生成动态SQL。 在MyBatis 3.2或以后版本，这个SqlBuilder废弃了，用了一个新的SQL类替代。 如果是false，MBG会使用新的SQL生成器生成客户端代码。  *默认值是 false* |

**示例**

此元素指定我们总希望生成的接口和对象在"text.model'包中，并且我们希望使用基于schema和catalog的子包。 它还指定了我们想使用为MyBatis3的XML配置文件的mapper接口。

<javaClientGenerator targetPackage="test.model"

targetProject="\MyProject\src" type="XMLMAPPER">

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

</javaClientGenerator>

**The <javaModelGenerator> Element**

The <javaModelGenerator> element is used to define properties of the Java model generator. The Java Model Generator builds primary key classes, record classes, and Query by Example classes that match the introspected table. This element is a required child element of the [<context>](http://mbg.cndocs.tk/configreference/context.html) element.

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| targetPackage | This is the package where the generated classes will be placed. In the default generator, the property "enableSubPackages" controls how the actual package is calculated. If true, then the calculated package will be the targetPackage plus sub packages for the table's catalog and schema if they exist. If false (the default) then the calculated package will be exactly what is specified in the targetPackage attribute. MyBatis Generator will create folders as required for the generated packages. |
| targetProject | This is used to specify a target project for the generated objects. When running in the Eclipse environment, this specifies the project and source folder where the objects will be saved. In other environments, this value should be an existing directory on the local file system. MyBatis Generator will not create this directory if it does not exist. |

**可选属性**

无

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)

**支持的属性**

下面的表格列出了所有可用的 [<property>](http://mbg.cndocs.tk/configreference/property.html) 子元素:

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| constructorBased | This property is used to select whether MyBatis Generator will generate a constructor for the class that accepts a value for each field in the class. Also, the SQL result map will be built to use the constructor rather than the "setter" for each field.  This property is only applicable for MyBatis3 and will be ignored for iBATIS2.  This property is ignored (and forced "true") if the "immutable" property is set "true".  This property can be overridden by the corresponding property in a [<table>](http://mbg.cndocs.tk/configreference/table.html) element.  *默认值是 false* |
| enableSubPackages | This property is used to select whether MyBatis Generator will generate different Java packages for the objects based on the catalog and schema of the introspected table.  For example, suppose a table MYTABLE in schema MYSCHMA. Also suppose that the targetPackage attribute is set to "com.mycompany". If this property is true, the generated objects for the table will be placed in the package "com.mycompany.myschema". If the property is false, the generated objects will be placed in the "com.mycompany" schema.  *默认值是 false* |
| immutable | This property is used to select whether MyBatis Generator will generate immutable model classes - this means that the classes will not have "setter" methods and the constructor will accept values for each field in the class.  If true, this forces the model classes to be built with paramterized constructors regardless of the value of the "constructorBased" property.  This property is only applicable for MyBatis3 and will be ignored for iBATIS2.  This property can be overridden by the corresponding property in a [<table>](http://mbg.cndocs.tk/configreference/table.html) element.  *默认值是 false* |
| rootClass | This property can be used to specify a root class for all generated Java model objects. MyBatis Generator will specify this value as the super class of the primary key object, if the table has a primary key, or the record object otherwise. This value may be overridden by specifying the rootClass property on a Table configuration.  **重要:** If MyBatis Generator is able to load the root class, then it will not override a property in the root class that exactly matches a property that would normally be generated. An exact match of a property is defined as follows   * Property name matches exactly * Property is of the same type * Property has a "getter" method * Property has a "setter" method   If specified, the value of this property should be a fully qualified class name (like com.mycompany.MyRootClass). |
| trimStrings | This property is used to select whether MyBatis Generator adds code to trim the white space from character fields returned from the database. This can be useful if your database stores data in CHAR fields rather than VARCHAR fields. When true, MyBatis Generator will insert code to trim character fields.  *默认值是 false* |

**示例**

This element specifies that we always want to place generated classes in the "'test.model" package and that we want to use subpackages based on the table schema and catalog. We also want String columns trimmed.

<javaModelGenerator targetPackage="test.model"

targetProject="\MyProject\src">

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

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

</javaModelGenerator>

**The <javaTypeResolver> Element**

The <javaTypeResolver> element is used to define properties of the Java Type Resolver. The Java Type Resolver is used to calculate Java types from database column information. The default Java Type Resolver attempts to make JDBC DECIMAL and NUMERIC types easier to use by substituting Integral types if possible (Long, Integer, Short, etc.) If this behavior is undesirable, set the property "forceBigDecimals" to "true". You can also substitute your own implementation if you want different behavior than the default. This element is an optional child element of the [<context>](http://mbg.cndocs.tk/configreference/context.html) element.

**必选属性**

无

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| type | This can be used to specify a user provided Java Type Resolver. The class must implement the interface org.mybatis.generator.api.JavaTypeResolver, 必须有一个公开默认的构造函数。. The attribute also accepts the special value DEFAULT in which case the default implementation will be used (this has the same effect as not specifying the type). |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)

**支持的属性**

下面的表格列出了所有可用的 [<property>](http://mbg.cndocs.tk/configreference/property.html) 子元素:

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| forceBigDecimals | This property is used to specify whether MyBatis Generator should force the use of java.math.BigDecimal for DECIMAL and NUMERIC fields, rather than substituting integral types when possible. 这个属性有以下可选值：   |  |  | | --- | --- | | **false** | *这是默认值*  当这个属性是false或者没有指定时， the default Java type resolver will attempt to make JDBC DECIMAL and NUMERIC types easier to work with by substituting Integral types if possible. The substitution rules are as follows:   * If the scale is greater then zero, or the length is greater than 18, then the java.math.BigDecimal type will be used * If the scale is zero, and the length is 10 through 18, then the Java type resolver will substitute a java.lang.Long. * If the scale is zero, and the length is 5 through 9, then the Java type resolver will substitute a java.lang.Integer. * If the scale is zero, and the length is less than 5, then the Java type resolver will substitute a java.lang.Short. | | **true** | 当这个属性是true时， the Java type resolver will always use java.math.BigDecimal if the database column is of type DECIMAL or NUMERIC. | |

**示例**

This element specifies that we always want to use the java.math.BigDecimal type for DECIMAL and NUMERIC columns:

<javaTypeResolver>

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

</javaTypeResolver>

**<jdbcConnection> 元素**

<jdbcConnection> 元素用于指定数据库连接所需的内省表的属性。 MyBatis Generator 使用JDBC的数据库元数据类来发现您在配置中指定的属性表。 每一个[<context>](http://mbg.cndocs.tk/configreference/context.html)都必须有一个<jdbcConnection>元素。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| driverClass | 用于访问数据库的JDBC驱动程序的完全限定类名称。 |
| connectionURL | 用于访问数据库的JDBC连接URL。 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| userId | 访问数据库的用户ID。 |
| password | 访问数据库的密码。 |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N) 注意: 这里指定的任何属性将添加到JDBC驱动程序的属性中。

**示例**

此元素将连接到DB2数据库配置为MBGTEST在DB2客户端配置实用程序。 使用JDBC驱动程序的默认安装位置:

<jdbcConnection driverClass="COM.ibm.db2.jdbc.app.DB2Driver"

connectionURL="jdbc:db2:MBGTEST"

userId="db2admin"

password="db2admin">

</jdbcConnection>

**<plugin> 元素**

<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。 这是 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素的一个子元素。 可以在context指定任意数量的插件。插件将在配置中配置的顺序执行。

有关执行插件的详细信息，请参阅 [Implementing Plugins](http://mbg.cndocs.tk/reference/pluggingIn.html) 页面的引用。

有关MyBatis Generator提供的插件代码生成器的详细信息，请参阅 [Supplied Plugins](http://mbg.cndocs.tk/reference/plugins.html)提供参考页。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| type | 实现该接口的类的完全限定名的插件。该类必须实现该接口 org.mybatis.generator.api.Plugin, 必须有一个公开默认的构造函数。注意，继承 org.mybatis.generator.api.PluginAdapter 这个适配器类比继承接口更容易扩展。 |

**可选属性**

无

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0 or 1)

# <properties> 元素

<properties> 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。 属性配置中的任何属性将接受 ${property} 这种形式的属性。 从指定的属性文件中搜索匹配的值，配置的值将会被替换。 属性文件时正常的Java属性文件的格式。

<properties>元素是 [<generatorConfiguration>](http://mbg.cndocs.tk/configreference/generatorConfiguration.html) 元素的子元素。

## 必选属性

以下属性是必需的，并且只能有其中一个。

|  |  |
| --- | --- |
| **属性** | **描述** |
| resource | 属性文件的全限定名称。当指定了resource属性，将会从classpath下面搜索属性文件。 当指定为 com/myproject/generatorConfig.properties 时，他必须存在 com.myproject 包下面。 |
| url | 属性文件的URL值。这可以用于指定一个属性文件在文件系统上的特定位置，例如 file:///C:/myfolder/generatorConfig.properties。 |

## 可选属性

无

## 子元素

无

# <property> 元素

<property> 元素用于指定许多其他元素的属性。 每个支持属性元素的文档页面列出的不同值都是有效的。 此元素也可以用于将属性传递给您的任何自定义的代码生成器执行。

## 必选属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | 属性的名称(区分大小写)。 |
| value | 属性的值(通常是不区分大小写)。 |

## 可选属性

无

## 子元素

无

**<sqlMapGenerator> 元素**

<sqlMapGenerator> element 用于定义SQL映射生成器的属性。 SQL映射生成器为每一个内省的表生成MyBatis/iBATIS形式的XML文件。

如果目标是iBATIS2，那么这个元素是[<context>](http://mbg.cndocs.tk/configreference/context.html) 元素的一个必须的子元素。

如果目标是MyBatis3，那么只有当您选择javaClientGenerator需要XML时，他才是 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素的一个必须的子元素。

如果不指定一个javaClientGenerator，则适用以下规则:

* 如果您指定一个sqlMapGenerator，那么MBG将只生成SQL映射的XML文件和模型对象。
* 如果您没有指定一个sqlMapGenerator，那么MBG将只会生成模型对象。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| targetPackage | 这个包用于放置生成的SQL映射文件。 在默认的生成器中，属性"enableSubPackages"用来控制如何计算实际的包。 如果是true，如果表的catalog和schema存在，就将他们作为子包加起来。 如果"enableSubPackages"是false（默认值），计算的package将是targetPackage属性指定的值。 MBG 将创建所需的生成包的文件夹。 |
| targetProject | 这用来指定生成SQL映射的目标项目。 当在Eclipse环境中运行时，此选项指定保存对象的位置的项目和源文件夹。 在其他环境中，此值应为本地文件系统上的现有目录。 如果它不存在，MBG不会创建它。 |

**可选属性**

无

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)

**支持的属性**

下面的表格列出了所有可用的 [<property>](http://mbg.cndocs.tk/configreference/property.html) 子元素:

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| enableSubPackages | 这个属性用来选择MGB是否根据基于目录和内省表来生成不同的Java包。  例如，假设某个表MYTABLE在MYSCHMA的schema中，也假设targetPackage属性的值设置为"com.mycompany"。 如果此属性为true，为这个表生成的DAO接口和类将被放置在包"com.mycompany.myschema"中。 如果此属性是false，生成的SQL映射将被放在"com.mycompany" schema中。  *默认值是 false* |

**示例**

此元素指定我们总希望生成的SQL映射文件在"text.model'包中，并且我们希望使用基于schema和catalog的子包。 它还指定了我们想使用为MyBatis3的XML配置文件的mapper接口。

<sqlMapGenerator targetPackage="test.model"

targetProject="\MyProject\src">

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

</sqlMapGenerator>

**<table> 元素**

<table> 元素用来从内省数据库中选择一个表。选定的表会生成如下的这些对象：

* 一个 MyBatis/iBATIS 格式的SQL映射文件
* 一组根据表生成的"model"类:
  + 一个和表的主键匹配的类(如果表有主键)。
  + 一个包含了非主键字段和BLOB字段的类，如果存在主键类，这个类就会继承主键类。
  + 如果表中包含BLOB字段，就会有一个包含了BLOB字段的类。这个类会根据表的配置继承上面的一个或两个类。
  + 一个用于不同的"by example"方法(selectByExample, deleteByExample)生成动态查询条件的类。
* (可选) 一个DAO接口或者类

这个元素是[<context>](http://mbg.cndocs.tk/configreference/context.html)元素的一个至少存在一个的必选子元素。 您可以指定不限制数量的table元素。

**数据库标识符**

MyBatis Generator (MBG) 试图自动处理数据库标识符的大小写敏感性。 在大多数情况下，无论您是否设置catalog, schema 和 tableName 属性，MBG都能找到表。 MBG的处理过程遵循以下步骤：

1. 如果 catalog, schema 或 tableName 属性包含空格，MBG将会根据指定的精确的情况查找。 在这种（包含空格）情况下，MBG将会自动分割生成SQL中表的标识符。
2. 如果数据库记录是以大写形式存储的标识符，MGB会自动将任何表的标识符转换为大写。
3. 如果数据库记录是以小写形式存储的标识符，MGB会自动将任何表的标识符转换为小写。
4. 其他情况，MBG根据指定的精确的情况查找。

在大多数情况下，这种处理很完美。然而，在某些情况下它会失败。 例如，假设您创建了这样的一个表：

create table "myTable" (

...some columns

)

因为表名可以用分隔符，即使数据库把标识符存储为大写，大部分的数据库会根据精确指定的表名来生成表。 在（上面）这种情况下，您需要在表的配置中指定delimitIdentifiers="true"。

**必选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| tableName | 数据库表的名称(不包括schema或catalog)。如果需要，指定的值可以包含SQL通配符。 |

**可选属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| schema | 数据库 schema - 如果您的数据库不使用 schema ，或者有一个默认的 schema 您不需要设置 schema。 如果需要，指定的值可以包含SQL通配符。 |
| catalog | 数据库 catalog - 如果您的数据库不使用 catalog，或者有一个默认的 catalog，您就不需要设置 catalog。 |
| 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。 这个（指定或者自动生成的）名字将用于计算实体类的名称和DAO类的名称。  您可以在实体对象的名字上指定包名分割。 例如，您可以指定foo.Bar，然后实体对象的名字会是 Bar，包名foo会添加到生成器配置中指定的目标包后面。 |
| enableInsert | 指定是否生成Insert语句。  默认值是 *true*。 |
| enableSelectByPrimaryKey | 指定是否生成通过主键查询的语句。 无论这个怎么设置，当表不存在主键的时候，不会生成这个语句。  默认值是 *true*。 |
| enableSelectByExample | 指定是否生成通过Example查询的语句。 这个语句支持运行时生成多种不同条件的动态查询。  默认值是 *true*。 |
| enableUpdateByPrimaryKey | 指定是否生成通过主键更新的语句。 无论这个怎么设置，当表不存在主键的时候，不会生成这个语句。  默认值是 *true*。 |
| enableDeleteByPrimaryKey | 指定是否生成通过主键删除的语句。 无论这个怎么设置，当表不存在主键的时候，不会生成这个语句。  默认值是 *true*。 |
| enableDeleteByExample | 指定是否生成通过Example删除的语句。 这个语句支持运行时生成多种不同的条件动态删除。  默认值是 *true*。 |
| enableCountByExample | 指定是否生成通过Example查询总数的语句。 这个语句将返回满足Example条件的数据总数。  默认值是 *true*。 |
| enableUpdateByExample | 指定是否生成通过Example更新的语句。 这个语句将更新满足Example条件的数据。 如果设置为True,UpdateByExampleSelective语句也会生成。 这个语句只会更新那些参数中值不为null的的列。  默认值是 *true*。 |
| selectByPrimaryKeyQueryId | 这个值会以"'<value>' as QUERYID"这种形式被添加到通过主键查询的语句的select列中。 这可能对在运行时的DBA跟踪工具中标记查询有用。 如果您使用这个值，您需要为MBG生成的每一个查询指定一个唯一的id。 |
| selectByExampleQueryId | 这个值会以"'<value>' as QUERYID"这种形式被添加到通过Example查询的语句的select列中。 这可能对在运行时的DBA跟踪工具中标记查询有用。 如果您使用这个值，您需要为MBG生成的每一个查询指定一个唯一的id。 |
| modelType | 如果您需要，这个值可以用来重写默认的模型类型。 如果没有指定，MBG将会生成基于上下文默认模型类型的实体对象。 模型类型定义了MBG如何生成实体类。 有一些模型类型MGB会为每个表生成一个单独的实体类。 另外一些模型，MGB会根据表结构生成不同的一些类。 这个属性有以下可选值：   |  |  | | --- | --- | | **conditional** | 这个模型和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。 | | **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（注：BLOB类 继承 其他字段类 继承 主键类）。 | |
| 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 | 这个属性表示当查询表并且在生成的SQL中分隔标识符时，是否使用指定的确切的值。 有关更多详细信息，请参见上面的详述。  分隔符在 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素上指定。  默认值是 *false*。除非 catalog, schema 或 tableName 属性值包含空白时，是 *true*. |
| delimitAllColumns | 指示是否给生成SQL中所有的列名添加分隔符。 这是一种给每个列添加<columnOverride>来指定列需要被分隔的替代方式。 这对类似PostgreSQL这种使用小写标识符的数据库很有用。  分隔符在 [<context>](http://mbg.cndocs.tk/configreference/context.html) 元素上指定。  默认值是 *false*. |

**子元素**

* [<property>](http://mbg.cndocs.tk/configreference/property.html) (0..N)
* [<generatedKey>](http://mbg.cndocs.tk/configreference/generatedKey.html) (0 or 1)
* [<columnRenamingRule>](http://mbg.cndocs.tk/configreference/columnRenamingRule.html) (0 or 1)
* [<columnOverride>](http://mbg.cndocs.tk/configreference/columnOverride.html) (0..N)
* [<ignoreColumn>](http://mbg.cndocs.tk/configreference/ignoreColumn.html) (0..N)

**支持的属性**

下面的表格列出了所有可用的 [<property>](http://mbg.cndocs.tk/configreference/property.html) 子元素:

|  |  |
| --- | --- |
| **属性名** | **属性值** |
| constructorBased | 此属性用于选择代码生成器是否生成接受类中的每个字段的值的类的构造函数。 此外，SQL结果映射也会为每个字段生成使用构造方法而不是"setter"的方式。  此属性仅适用于MyBatis3，iBATIS2将会被忽略。  如果"immutable"属性设置为"true"，这个属性将会被忽略。  *默认值是 false* |
| ignoreQualifiersAtRuntime | 如果设置为true，MBG不会讲schema或catalog添加到生成SQL中的表名上。 如果您有几个schemas中的表具有相同的名称，这是非常有用的。 您可以使用MBG生成基于在一个schema上的表，但是运行时不包含schema。  *默认值是 false* |
| immutable | 此属性用于选择MBG是否会生成不可变的模型类。 这意味着这些类不会包含"setter"方法，而且通过构造参数接收类中所有字段的值。  如果设置为true,将会忽略"constructorBased"属性， 强制生成通过参数化构造方法的模型类。  此属性仅适用于MyBatis3，iBATIS2将会被忽略。  *默认值是 false* |
| modelOnly | 此属性用于选择MBG是否只会为表生成模型类。  如果设置为true，那么就不会生成Java客户端类。 如果 <sqlMapGenerator> 配置了，并且属性设置为true，那么MGB将会在该表的SQL映射XML中只生成结果映射元素。  如果设置为true，这个值会覆盖<table>元素上所有的"enable\*"属性，将不会生成任何CRUD方法。  *默认值是 false* |
| rootClass | 这个属性可以用来指定所有生成的Java模型类的基类。 如果表包含主键，MBG会将该值指定为主键对象的超类。 或者其他记录对象的超类。 这个值将会覆盖Java模型生成配置中配置的rootClass  **重要:** 如果MBG可以加载基类，通常生成的属性不会覆盖和基类中完全匹配的属性。 完全匹配属性的定义如下   * 属性名称完全相同 * 属性的类型相同 * 属性有一个"getter"方法 * 属性有一个"setter"方法   如果指定这个值，这个属性值应该是一个完全限定的类名(例如：com.mycompany.MyRootClass). |
| rootInterface | 这个属性可以用来指定所有生成的DAO接口对象的父接口。 这个属性值会覆盖DAO生成器配置中配置的rootInterface属性。  **重要:** MBG 不会校验接口是否存在或者合法。  如果指定这个值，这个属性值应该是一个完全限定的接口名 (例如：com.mycompany.MyRootInterface). |
| runtimeCatalog | 如果您指定了这个属性值，MBG会在生成的SQL中使用这个catalog，而不是前面配置的catalog 属性值。 当您生成代码的catalog和运行时的catalog不一样时，这会非常有用。 |
| runtimeSchema | 如果您指定了这个属性值，MBG会在生成的SQL中使用这个schema，而不是前面配置的schema 属性值。 当您生成代码的schema和运行时的schema不一样时，这会非常有用。 |
| runtimeTableName | 如果您指定了这个属性值，MBG会在生成的SQL中使用这个表名，而不是前面配置的tableName 属性值。 当您想在Oracle中使用公共同义词生成对象时，这会非常有用。 在这种情况下，您将需要生成对象的同义词的指向您实际表。那就在此属性中指定同义词名称。 在使用公共同义词的大多数情况下，您还需要设置 ignoreQualifiersAtRuntime属性。 |
| selectAllOrderByClause | 这个属性可以用于指定将会加到 selectAll 方法中的order by语句。 这仅适用于您使用MyBatis3Simple目标运行环境时。 MBG会将您在这儿指定的任何属性值前面追加order by， 因此该属性赢仅包含列表中的列(例如 ID1, ID2 或 ID1 desc, ID2 asc) |
| 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 <columnOverride> 元素显示指定，在这种情况下将会忽略这个（useActualColumnNames）属性。  例如，假设一个表包含START\_DATE列，如果这个属性是"true", 那么MBG生成的属性名是START\_DATE - 这意味着这个值的getter和setter方法将会是 getSTART\_DATE() 和 setSTART\_DATE(). 如果这个属性值是fasle,MBG将会生成的属性名是 startDate - 这意味着这个值的getter和setter方法将会是 getStartDate() 和 setStartDate().  *默认值是 false* |
| 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. 当表中的列名的区别只是大小写的时候,这会非常有用. 这个支持还会有一个轻微的性能优势.  *默认值是 false*  **重要提示:** 当目标运行环境是Mybatis版本3时不支持这个属性. |
| 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.  *默认值是 false* |

**示例**

这个元素指定我们总是想为一个在MYSCHEMA schema上的叫做 MYTABLE 的表生成代码. 我们还想忽略表中一个叫"fred"的列,而且我想还想重写"BEG\_DATA"列,以便生成的属性名是"startDate".

<table tableName="MYTABLE" schema="MYSCHEMA">

<ignoreColumn column="fred"/>

<columnOverride column="BEG\_DATE" property="startDate"/>

</table>

**使用生成的对象**

MyBatis Generator (MBG) 生成这些类型的对象:

1. [Java模型对象](http://mbg.cndocs.tk/generatedobjects/javamodel.html) (通常)
2. [SQL映射文件](http://mbg.cndocs.tk/generatedobjects/sqlmap.html) (iBATIS通常有, MyBatis经常有)
3. [SQL映射文件](http://mbg.cndocs.tk/generatedobjects/javaclient.html) (可选的)
4. 一个在xxxByExample方法中使用的类。有关这些类的信息，请参阅下列网页：
   * [Example类用法说明](http://mbg.cndocs.tk/generatedobjects/exampleClassUsage.html)
   * [扩展Example类](http://mbg.cndocs.tk/generatedobjects/extendingExampleClass.html)

这些独立的页面有这些对象的说明和用法。

**JAVA实体对象**

MyBatis Generator (MBG)根据数据库表字段生成Java类. MBG生成的是简单实体对象而非带业务逻辑的实体对象(详见[设计理念](http://mbg.cndocs.tk/philosophy.html)介绍).MBG根据表特性和配置生成不同类型的实体对象.

MBG生成字段和方法时包含JavaDoc标签 @mbggenerated.运行Eclipse插件时,MBG会将每个Java对象中的JavaDoc标签删除或替换.其他地方不会发生变化. 在这种情况下,您在Java类中添加的字段和方法就不用担心丢失--不包括包含了JavaDoc@mbggenerated标签部分内容.

离开Eclipse插件，您需要手动合并Java文件,但是使用JavaDoc的@mbggenerated标签用来了解删除以前版本的文件是否安全.

以下描述生成不同的实体对象.MBG根据[<context>](http://mbg.cndocs.tk/configreference/context.html)的defaultModelType属性(支持conditional、flat、hierarchical三个值)和[<table>](http://mbg.cndocs.tk/configreference/table.html)的modelType(此属性用于覆盖默认的defaultModelType)属性配置生成不同的实体对象.

任何字段配置了[<ignoreColumn>](http://mbg.cndocs.tk/configreference/ignoreColumn.html),自动生成时不会添加到Java对象中.

注意: 下面描述中"BLOB"指任何列的BLOB数据类型包括BLOB, CLOB, LONGVARCHAR, 和 LONGVARBINARY.

**主键类**

主键类在表中包含一个主键的字段属性. (MBG)根据表列明自动生成属性名称.自动生成的属性名称可以配置<columnOverride>属性覆盖.

类名默认情况是«TableName»Key,如果<table>配置了domainObjectName属性那么类名是«domainObjectName»Key.

如果表中存在一个字段的主键将会生成hierarchical类型实体对象. 如果表中存在多列组合主键将会生成conditional类型实体对象.主键类是不会生成flat类型实体对象 model.

**记录类**

主键类在表中不包含主键、BLOB的字段属性. 如果只有一个主键记录类将继承主键类. (MBG)根据表列明自动生成属性名称.自动生成的属性名称可以配置<columnOverride>属性覆盖.

类名默认情况是«TableName»,如果<table>配置了domainObjectName属性那么类名是«domainObjectName»配置的值.

如果表中不存在组合主键、BLOB列将生成hierarchical类型实体对象.如果表中不存一个字段的主键、BLOB列或只有一个主键或者一个BLOB列将生成conditional类型实体对象.记录类常常生成flat类型实体对象.

**BLOB记录类**

BLOB记录类在表中包含BLOB字段属性. 如果表中只存在一个字段该类将继承基础类, 或将继承主键类(注意:MBG不支持表中只包含BLBO列). (MBG)根据表列明自动生成属性名称.自动生成的属性名称可以配置<columnOverride>属性覆盖.

BLOB记录类调用selectByPrimaryKey或selectByExampleWithBLOBs方法返回BLOB字段值.

类名默认情况是«TableName»WithBLOBs,如果<table>配置了domainObjectName属性那么类名是«domainObjectName»WithBLOBs.

如果表中存在一个BLOB列将生成hierarchical类型实体对象. 如果表中存在多个BLOB列将生成conditional类型实体对象.BLOB记录类是不会生成flat类型实体对象.

**Example类**

Example类用来处理MBG动态查询功能. Example类设置条件用于动态WHERE子句下列方法中:

* selectByExample
* selectByExampleWithBLOBs
* deleteByExample
* countByExample
* updateByExample

Example类不继承任何实体对象.

类名默认情况是«TableName»Example,如果<table>配置了domainObjectName属性那么类名是«domainObjectName»Example.

如果方法被启动Example类将生成任何\*ByExample方法.注意:如果表中有非常多的字段该类可能很大,但 DAO生成的XML是比较小的. 如果您不需要使用动态WHERE子句,您可以禁用生成这些方法.

详见[Example类使用说明](http://mbg.cndocs.tk/generatedobjects/exampleClassUsage.html)了解详情.

**SQL映射文件**

MyBatis Generator (MBG)生成SQL映射文件遵循MyBatis或iBATIS SQL Map DTD规范. 在表的基础上这些文件还包含了不同的标签和属性配置. MBG按照配置表生成SQL映射文件.表名就是SQL映射文件的命名空间(前提是数据库支持schema和catalog,主流数据库对schema和catalog都支持).MBG不会自动把 SQL映射文件到MyBatis/iBATIS配置文件中 - 您必须手动把xxxMapper.xml文件加入到配置文件中(或者您使用一个插件使MBG生成配置文件;Spring配置文件可以实现自动扫描xxxMapper.xml和接口).

每个自动生成的XML方法标签下都包含@mbggenerated的注释块.在运行的时候@mbggenerated注释块将删除和替换.其他部分内容将保持不变. 考虑到这点,不用担心在运行时丢失您新增内容--不包括@mbggenerated块内容.

以下部分描述了将要生成的元素.

注意:下面描述中"BLOB"指任何列的BLOB数据类型包括BLOB, CLOB, LONGVARCHAR, 和 LONGVARBINARY.

**结果集**

结果集用于数据库表列映射Java对象的属性. 结果集(和相应的查询语句)不包括如下情况:

* 任何列配置<ignoreColumn>属性将被会被忽略
* 任何BLOB字段(详见有关BLOBs字段相关映射)

对于列配置<columnOverride>属性将映射成配置的名称.没有配置使用的是默认的属性和JDBC类型.

对于自定义连接查询结果集继承一个结果集是一种非常常见的用法. 对于其他连接查询也想使用该结果集在MBG自动生成的时候需要配置前缀.详见[<table>](http://mbg.cndocs.tk/configreference/table.html)属性alias配置前缀,这样做法可以区分表中相同字段.

如果table配置enableSelectByExample、enableSelectByPrimaryKey属性为true,MBG会生成结果集,table默认这两个配置都是true,所以大部分情况下都会生成resultMap.

**BLOB结果集**

表中存在BLOB字段时MBG会自动生成一个BLOB结果集,BLOB结果集继承了除BLOB字段外的基础结果集. 我们提供不同的版本查询语句,来适应查询结果中是否需要BLOB字段.

BLOB结果集(和相应的查询语句)不包括如下情况:

* 任何列配置<ignoreColumn>属性将被会被忽略

对于列配置<columnOverride>属性将映射成配置的名称.没有配置使用的是默认的属性和JDBC类型.

对于自定义连接查询结果集继承一个结果集是一种非常常见的用法.对于其他连接查询也想使用该结果集,在MBG自动生成的时候需要配置前缀.详见[<table>](http://mbg.cndocs.tk/configreference/table.html) 性alias配置前缀,这样做法可以区分表中相同字段.

如果table存在BLOB字段且配置了enableSelectByExample、enableSelectByPrimaryKey属性,MBG会生成BLOB结果集,table默认这两个配置都是true所以大部分情况下都会生成resultMap.

**Where条件SQL语句**

"by example"方法重用Where条件.被重用的Where条件不包括BLOB字段.大部分表WHERE条件都不支持BLOB字段.

"by example"语句被配置将动态生成Where条件语句.

**根据主键查询**

select语句按照主键查询返回一行数据.如果表中存在BLOB字段,查询结果中会包含该字段.

如果table配置了enableSelectByExample、enableSelectByPrimaryKey属性为true,MBG会生成结果集,table默认这两个配置都是true所以大部分情况下都会生成resultMap.

**根据条件查询**

根据条件查询结果集和自动生成对象相匹配. 通过实现一个简单的"query by example"方法可以适用于不同的数据库查询.就算表中存在BLOB字段该查询结果集也不会返回该BLOB字段 (具体返回BLOB字段详见根据条件查询包含BLOB字段结果集部分描述).

**重要:**条件为空或者没有设置条件时,会将表中**所有**数据查询出来.

如果table配置了enableSelectByExample属性为true,MBG会生成Example查询.

**根据条件查询包含BLOB字段(方法:selectByExampleWithBLOBs)**

Example查询结果集和自动生成对象相匹配. 通过实现一个简单的"query by example"方法可以适用于不同的数据库查询.算表中存在BLOB字段该查询结果集将返回该BLOB字段.

**重要:** 查询条件为空或者没有设置查询条件时,会将表中**所有**数据查询出来.

如果table配置了enableSelectByExample属性为true,MBG会生成包含BLOB字段的Example查询.

**插入**

该方法可以插入表中所有字段(包括BLOB字段),但列配置了<ignoreColumn>属性将不会插入.

如果表有主键自动增长(自动增长列或序列),table子元素配置<generatedKey>属性,MBG会生成<selectKey>块语句,一般情况下都不需要配置使用数据库自动增长即可,除非表字段需要使用序列.

**重要提示:**与iBATIS2和MyBatis3插入的差异.差异如下:

|  |  |
| --- | --- |
| **运行** | **特性** |
| iBATIS2 | 配置了<generatedKey>属性,插入的时候会返回新的自动增长值.没有配置<generatedKey>属性, 插入返回值为void. |
| MyBatis3 | 插入方法将返回插入受影响的行数(通常是0或者1). 配置了<generatedKey>属性,自动增长值会被设置在参数对象中返回. |

如果table配置了enableInsert属性为true,MBG会生成该方法.

**选择性插入**

该方法可以插入表中所有字段(包括BLOB字段),但列配置了<ignoreColumn>属性将不会插入.然而,参数对象为空的列将不会插入值.如果数据库列设置了默认值则取默认值,否则列值是空. 对于列不允许为空情况-则需要人为确保调用该方法时不为空列确保有值使插入不报错.**重要:**任何映射到java引用类型的字段都会被该方法插入.

如果表有主键自动增长(自动增长列或序列),table子元素配置<generatedKey>属性,MBG会生成<selectKey>块语句,一般情况下都不需要配置使用数据库自动增长即可,除非表字段需要使用序列.

**重要提示:**与iBATIS2和MyBatis3插入的差异.差异如下:

|  |  |
| --- | --- |
| **运行** | **特性** |
| iBATIS2 | 配置了<generatedKey>属性,插入的时候会返回新的自动增长值.没有配置.没有配置<generatedKey> 属性, 插入返回值为void. |
| MyBatis3 | 插入方法将返回插入受影响的行数(通常是0或者1). I配置了<generatedKey>属性,自动增长值会被设置在参数对象中返回. |

如果table配置了enableInsert属性为true,MBG会生成该方法.

**根据主键更新**

该方法按照主键修改一行数据.该方法会修改表中所有字段,除非:

* 列配置了<ignoreColumn>属性
* BLOB字段(详见根据主键更新包含BLOB字段)

数据库表中存在主键且table配置了enableUpdateByPrimaryKey属性为true,MBG会生成该方法.

**根据主键更新包含BLOB字段**

该方法按照主键修改一行数据.该方法会修改表中所有字段(包括BLOB字段),除非:

* 列配置了<ignoreColumn>属性

数据库表中存在主键包含BLOB列且table配置了enableUpdateByPrimaryKey属性为true,MBG会生成该方法.

**根据主键选择性更新**

该方法按照主键修改一行数据.该方法会更新表中参数对象属性不为空的列.该方法可以被使用更新某些特定的列而不影响所有列数据. **重要:** 任何映射到java引用类型的字段都会被该方法更新.

数据库表中存在主键且table配置了enableUpdateByPrimaryKey属性为true,MBG会生成该方法.

**根据主键删除记录**

该方法按照主键删除一行数据.

数据库表中存在主键且table配置了enableDeleteByPrimaryKey属性为true,MBG会生成该方法.

**根据条件删除记录**

该方法按照条件将删除一行或者多行数据.

**重要:**条件为空或者没有设置条件时, 会将表中**所有**记录删除.

如果table配置了enableDeleteByExample属性为true,MBG会生成该方法.

**根据条件查询记录总数**

该方法按照条件查询表中符合条件记录的记录总数.

**重要:**条件为空或者没有设置条件时,返回表中所有记录总数.

如果table配置了enableCountByExample属性为true,MBG会生成该方法.

**根据条件更新记录**

该方法按照条件更新表中符合条件记录.该方法会修改表中所有字段,除非:

* 列配置了<ignoreColumn>属性
* BLOB字段(详见根据条件修改记录包含BLOB字段)

**重要:**条件为空或者没有设置条件时, 会将表中**所有**记录更新.

如果table配置了enableUpdateByExample属性为true,MBG会生成该方法.

**根据条件修改记录包含BLOB字段**

该方法按照条件更新表中符合条件记录.该方法会修改表中所有字段(包括BLOB字段),除非:

* 列配置了<ignoreColumn>属性

**重要:**条件为空或者没有设置条件时, 会将表中**所有**记录更新.

如果表存在BLOB字段且table配置了enableUpdateByExample属性为true,MBG会生成该方法.

**根据条件选择性更新记录**

该方法按照条件更新表中符合条件记录.该方法会更新表中参数对象属性不为空的列.该方法可以被使用更新某些特定的列而不影响所有列数据.**重要:** 任何映射到java引用类型的字段都会被该方法更新.

**重要:**条件为空或者没有设置条件时, 会将表中**所有**记录更新.

如果table配置了enableUpdateByExample属性为true,MBG会生成该方法.

**Java客户端对象**

MyBatis Generator (MBG)生成几种类型的Java客户端对象.用Java的客户端对象与所产生的XML交互容易得多. 对于配置中的每个表，MBG生成一个或多个Java客户端对象.如MyBatis的3, 这些都接口映射.对于iBATIS的2.x中，这些都是DAO接口和实现类. 生成Java客户对象是可选的,由<javaClientGenerator>配置. MBGr可以产生以下的Java客户短对象类型:

* 如MyBatis 3.x:
  + XMLMAPPER - 用与支持MyBatis 3.x映射
* 如iBATIS 2.x:
  + IBATIS - 使用iBATIS DAO框架
  + SPRING - 与Spring框架联合使用
  + GENERIC-CI - 使用不依赖iBATIS数据映射
  + GENERIC-SI - 使用不依赖iBATIS数据映射

每个字段和方法产生包括JavaDoc标签@mbggenerated.Eclipse插件运行时, 运行中每个字段和方法包括此Javadoc标记将被删除并替换.在类中的其他任何字段或方法将保持不变. 考虑到这一点,您可以不用担心加入的字段和方法丢失 -不包括在Javadoc标签@mbggenerated中内容.

离开Eclipse插件,您需要手动合并Java文件,但是使用JavaDoc的 @mbggenerated标签用来了解删除以前版本的文件是否安全.

注意: 下面描述中"BLOB"指任何列的BLOB数据类型包括BLOB, CLOB, LONGVARCHAR, 和 LONGVARBINARY.

**通用DAO方法**

根据表的特性,以及配置选项,Java客户端自动生成如下方法:

* countByExample
* deleteByPrimaryKey
* deleteByExample
* insert
* insertSelective
* selectByPrimaryKey
* selectByExample
* selectByExampleWithBLOBs
* updateByPrimaryKey (否更新BLOB字段需要重写方法)
* updateByPrimaryKeySelective (只更新参数类非空字段)
* updateByExample (否更新BLOB字段需要重写方法)
* updateByExampleSelective (只更新参数类非空字段)

对于包含BLOB的表,MBG通过生成不同的对象和方法使您更容易使用BLOB字段,是否忽略它们,这取决于具体情况.

详见 [Example类使用说明](http://mbg.cndocs.tk/generatedobjects/exampleClassUsage.html) 中selectByExample方法使用.

**XMLMAPPER 客户端 (MyBatis 3.x)**

XMLMAPPER客户端是将接口方法映射到生成的XML映射文件中.例如,MBG自动生成的接口名为MyTableMapper.您可以如下使用该接口:

SqlSession sqlSession = sqlSessionFactory.openSession();

try {

MyTableMapper mapper = sqlSession.getMapper(MyTableMapper.class);

List<MyTable> allRecords = mapper.selectByExample(null);

} finally {

sqlSession.close();

}

有关如何创建实例的详细信息,请参阅MyBatis文档怎么创建sqlSessionFactory实例.

**IBATIS DAOs (iBATIS 2.x)**

iBATIS DAO依赖于iBATIS的DAO框架(iBATIS的一个可选部分 - 现在已经过时). 他们继承SqlMapDaoTemplate类和初始化DAOManager对象的实例,并且调用方法执行不同的语句.

MBG不会为您更新“dao.xml”文件 - 您必须添加手动相应的内容.

iBATIS DAO框架是一个非常基础的IoC容器,如果您尚未使用像Spring或PicoContainer的管理依赖关系的框架,iBATIS DAO框架可能会有用. 然而,该框架现在已经过时,我们建议您使用Spring.

**SPRING DAOs (iBATIS 2.x)**

SPRING DAO依赖于Spring框架.他们继承Spring的SqlMapClientDaoSupport类通过Spring容器构造DAO.

**GENERIC-CI DAOs (iBATIS 2.x)**

GENERIC-CI DAO调用iBATIS的SqlMapClient接口.该接口的实例是通过构造函数注入.

**GENERIC-SI DAOs (iBATIS 2.x)**

GENERIC-SI DAO调用iBATIS的SqlMapClient接口.该接口的实例是通过setter注入.

**Example类使用说明**

Example类指定如何构建一个动态的where子句. 表中的每个non-BLOB列可以被包括在where子句中. 例子是展示此类用法的最好方式.

Example类可以用来生成一个几乎无限的where子句.

Example类包含一个内部静态类 Criteria 包含一个用 anded 组合在where子句中的条件列表. Example类包含一个 List<Criteria> 属性,所有内部类Criteria中的子句会用 ored组合在一起. 使用不同属性的 Criteria 类允许您生成无限类型的where子句.

创建 Criteria 对象 可以使用Example类中的 createCriteria() 或者 or() . 如果 Criteria 对象是用 createCriteria() 创建的，它会自动为 List<Criteria> 属性添加一个 Criteria 对象 - 这使得它更容易写一个简单的where子句， 如果您不需要 or 或者其他几个子句组合的话. 用 or(Criteria criteria) 方法创建 Criteria 对象, 方法里的 criteria 对象会被添加进 Criteria 对象的列表中.

**重要** 我们推荐您只使用 or() 方法创建 Criteria 对象. 我们相信这种方法使代码更有可读性.

**简单查询**

这个例子展示了如何用生成后的Example类去生成一个简单的where子句:

TestTableExample example = new TestTableExample();

example.createCriteria().andField1EqualTo(5);

作为另一种选择, 下面的方式也是可以的:

TestTableExample example = new TestTableExample();

example.or().andField1EqualTo(5);

在上面的例子中, 动态生成的where子句是:

where field1 = 5

**复杂查询**

下面的例子展示了如何用生成后的Example类去生成一个复杂的where子句 (用到了 JSE 5.0 的泛型):

TestTableExample example = new TestTableExample();

example.or()

.andField1EqualTo(5)

.andField2IsNull();

example.or()

.andField3NotEqualTo(9)

.andField4IsNotNull();

List<Integer> field5Values = new ArrayList<Integer>();

field5Values.add(8);

field5Values.add(11);

field5Values.add(14);

field5Values.add(22);

example.or()

.andField5In(field5Values);

example.or()

.andField6Between(3, 7);

在上面的例子中, 动态生成的where子句是:

where (field1 = 5 and field2 is null)

or (field3 <> 9 and field4 is not null)

or (field5 in (8, 11, 14, 22))

or (field6 between 3 and 7)

将会返回满足这些条件的记录结果.

**去重复查询**

您可以在所有的Example类中调用 setDistinct(true) 方法进行强制去重复查询.

**Criteria类**

Criteria 内部类的每个属性都包含 andXXX 方法，以及如下的标准的SQL查询方法:

* IS NULL - 指相关的列必须为NULL
* IS NOT NULL - 指相关的列必须不为NULL
* = (equal) - 指相关的列必须等于方法参数中的值
* <> (not equal) - 指相关的列必须不等于方法参数中的值
* > (greater than) - 指相关的列必须大于方法参数中的值
* >= (greater than or equal) - 指相关的列必须大于等于方法参数中的值
* < (less than) - 指相关的列必须小于于方法参数中的值
* <= (less than or equal) - 指相关的列必须小于等于方法参数中的值
* LIKE - 指相关的列必须 "like" 方法参数中的值. 这个方法不用必须加入 '%', 您必须设置方法参数中的值.
* NOT LIKE - 指相关的列必须 "not like" 方法参数中的值. 这个方法不用必须加入 '%', 您必须设置方法参数中的值.
* BETWEEN - 指相关的列必须在 "between" 方法参数中的两个值之间.
* NOT BETWEEN - 指相关的列必须不在 "not between" 方法参数中的两个值之间.
* IN - 指相关的列必须在传入的方法参数的list中.
* NOT IN - 指相关的列必须不在传入的方法参数的list中.

# 扩展Example类

在某些情况下可能需要扩展自动生成的example。您可能希望添加特定数据库查询条件(如Oracle ROWNUM支持),或添加除自动生成外的查询条件(如不区分大小写查询)。在这种情况下，您需要扩展自动生成example类来添加这些额外的查询条件。

## 一般原则

MyBatis Generator (MBG)一般情况下一个表名对应一个自动生成"example"类,除非您特殊配置。"example"类动态生成where条件被用于xxxByExample方法。

标准的"example"类包含只是标准SQL条件查询功能。在这种情况下,程序特定需要您添加额外条件查询功能。这可能需要添加非标准条件查询或在where条件中使用数据库特定功能。

自动生成"example"类中包含一个内部类实现where条件查询功能。内部类命名为GeneratedCriteria。 MBG同时也生成了内部类Criteria继承了GeneratedCriteria,您可以使用它在example类中添加您想要的功能。Eclipse插件不会删除Criteria类新增代码(注：只有Eclipse插件,并且有注释和时间戳等要求才能自动合并),因此您无需担心新增代码丢失。

例如,有一个表叫CUSTOMER。通常,MBG生成一个名为 CustomerExample的类。在CustomerExample类中添加额外功能,需要在CustomerExample.Criteria类中新增方法。

## 扩展vs插件

如果您经常扩展自动生成类,写一个插件来实现该功能比手工编写扩展类代码更方便。下面(标题"单条件参数")的插件类能够完成单参数查询 org.mybatis.generator.plugins.CaseInsensitiveLikePlugin。

## 添加条件语句

MBG自动生成SQL在运行允许创建无限制where条件。为了完成这个,自动生成SQL支持四大类型条件语句。对应每种类型的SQL语句,GeneratedCriteria 内部有一个对应的方法用于添加一个动态的where条件。

### 1. 简单字符串替换

在使用这种类型的条件查询时不需要参数对象替换where条件中。例如:

FIRST\_NAME is null  
LAST\_NAME is not null

此条件语句GeneratedCriteria类方法是:

   addCriterion(String anyString)

其中"anyString"是字符串替换where子句。该方法适合任何类型。

例如,您想使用SOUNDEX函数完成"类似"搜索功能。在MySQL中,条件语句应该是:

SOUNDEX(FIRST\_NAME) = SOUNDEX('frod')

此种查询太复杂，可以考虑使用另外一个方法,这种简单字符串替换必须插入到where条件中。在内部类Criteria中添加如下方法:

public Criteria andFirstNameSoundsLike(String value) {

StringBuffer sb = new StringBuffer("SOUNDEX(FIRST\_NAME) = SOUNDEX('");

sb.append(value);

sb.append("')");

addCriterion(sb.toString());

return this;

}

下面代码是在selectByExample方法中使用了刚才新增的方法:

CustomerExample example = new CustomerExample();

Criteria criteria = example.createCriteria();

criteria.andFirstNameSoundsLike("frod");

List results = selectByExample(example);

这种方法可以添加任何条件语句到where子句中。然而,由于需要保证不同数据类型的正确(最明显的日期、时间和时间戳),所以最好使用参数替换。同时, 这样操作暴露过多的方法会导致SQL注入问题。如果可能,我们建议使用下面列出的方法之一。

### 2. 单条件参数

使用这种类型作为条件语句,一个参数替换where条件。例如

FIRST\_NAME = ?  
LAST\_NAME <> ?

自动生成Criteria类条件方法如下:

   addCriterion(String anyString,Object anyObject,String propertyName)

Where:

**anyString**

替换where条件参数子句,如:upper(FIRST\_NAME) like

**anyObject**

条件值

**propertyName**

条件列名,用于排除潜在错误。

该方法用于单一参数where条件。

例如,假设您想特定列不区分大小写查询,在MySQL中查询条件如下:

upper(FIRST\_NAME) like ?

此方法适合单个参数功能-一个参数一个参数值。将下面方法添加到ExtendedCriteria中:

public ExtendedCriteria andFirstNameLikeInsensitive(String value) {

addCriterion("upper(FIRST\_NAME) like",

value.toUpperCase(),"firstName");

return this;

}

下面代码是在selectByExample方法中使用了刚才新增的功能:

ExtendedExample example = new ExtendedExample();

ExtendedCriteria criteria = (ExtendedCriteria) example.createCriteria();

criteria.andFirstNameLikeInsensitive("fred%");

List results = selectByExample(example);

### 3. 列表条件

列表条件适用于where条件中多个值的情况。例如:

FIRST\_NAME IN (?,?,?)  
LAST\_NAME NOT IN (?,?,?,?)

由于包含了"in" and "not in"这样的标准查询条件,因此使用起来不太灵活。然而在Criteria类中您会发现有相应的方法,如下:

   addCriterion(String anyString,List listOfObjects,String propertyName)

Where:

**anyString**

替换where条件参数,如:FIRST\_NAME IN

**listOfObjects**

list对象值替换条件值(在list前有一个开始的括号，一个结束括号在list后)。

**propertyName**

条件列名,用户排除潜在错误。

### 4. Between条件

Between条件参数适用于where条件特定的格式。例如:

FIRST\_NAME BETWEEN ? AND ?  
LAST\_NAME NOT BETWEEN ? AND ?

由于包含了"between" and "not between"这样的标准查询条件,因此使用起来不太灵活。然而在Criteria类中您会发现有相应的方法,如下:

   addCriterion(String anyString,Object object1,Object object2,String propertyName)

Where:

**anyString**

替换where条件参数,如:FIRST\_NAME BETWEEN

**object1**

替换where条件第一个参数值(object1后会自带一个"and"连接词)。

**object2**

替换where条件第二个参数值(object2前会自带一个"and"连接词)。

**propertyName**

条件列名,用户排除潜在错误。

**使用注意事项**

不同数据库使用注意事项。

* [DB2](http://mbg.cndocs.tk/usage/db2.html)
* [MySql](http://mbg.cndocs.tk/usage/mysql.html)
* [Oracle](http://mbg.cndocs.tk/usage/oracle.html)
* [PostgreSQL](http://mbg.cndocs.tk/usage/postgresql.html)

请让我们知道您所使用的数据库,我们很乐意将信息添加到本节以供将来参考。

# DB2 使用注意事项

## LONG VARCHAR 字段

默认情况下MyBatis将LONG VARCHAR字段映射为java.lang.String类型, 数据库字段映射为 jdbcType="LONGVARCHAR"。这样映射会导致DB2在检索数据时发生错误。DB2 的LONG VARCHAR字段应该被映射为 java.lang.String类型，数据库类型应该是jdbcType="VARCHAR"。为了解决该问题, 使用<columnOverride>配置如下:

<table schema="DB2ADMIN" tableName="ALLTYPES" >

<columnOverride column="LONG\_VARCHAR\_FIELD" javaType="java.lang.String" jdbcType="VARCHAR" />

</table>

# MySql 使用注意事项

## 无符号类型

MySql支持有符号，无符号，数字类型字段。这些不是JDBC类型，因此MyBatis 生成器不能自动转换这种类型的字段。JAVA数据类型都是有符号的， 当使用无符号类型时这就会导致缺失精度。可以使用 <columnOverride> 解决MySql无符号数值类型的字段。下面是一个如何处理无符号bigint字段类型例子:

<table tableName="ALLTYPES" >

<columnOverride column="UNSIGNED\_BIGINT\_FIELD" javaType="java.lang.Object" jdbcType="LONG" />

</table>

您还必须自己强制将返回值转换为适当的类型(在上面这种情况下，需要转换为 java.math.BigInteger)。

# Oracle 使用注意事项

## 公共同义词

如果您想要一个表生成对象公共同义词，您实际生成的对象执行真正的表名-然后在运行的时候修改表名。MyBatis自动生成器支持这样。

例如，假设有一个公共表名为"FRED"指向表"HR.EMPLOYEES"。下面的配置是基于HR.EMPLOYEES生成的对象，但运行的时候SQL只会指向FRED:

<table schema="HR" tableName="EMPLOYEES">

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

<property name="runtimeTableName" value="FRED" />

</table>

## Oracle 长数据类型

Oracle的JDBC驱动会将LONG类型的列当成JDBC的LONGVARCHAR类型。由于Oracle驱动程序不支持，MyBatis将映射为CLOB。 因此在使用长数据类型时，您应该在生成器上配置一个列的重写将其映射到JDBC的VARCHAR类型。

**PostgreSQL 使用注意事项**

**区分大小写**

PostgreSQL所有的数据库标识符(表名,模式名,列明等)是区分大小写的。此外，PostgreSQL明显的偏好所有标识符使用小写字母。 如果您使用全部小写标识符的PostgreSQL，MyBatis Generator不需要任何额外的配置就会发现表并且编写正确的SQL。如果您使用混合或者大写模式，您需要适当配置MyBatis Generator:

* 当您的表或模式是大小写混合或全部大写时，使用delimitIdentifiers配置。
* 为每个大小写混合或大写的列可以指定 <columnOverride>元素来指定列， 或者您可以指定delimitAllColumns属性来指定所有的列。

例子1:

<table schema="HR" tableName="Employees"

delimitIdentifiers="true" delimitAllColumns="true"/>

例子2...

<table schema="HR" tableName="Employees" delimitIdentifiers="true" >

<columnOverride column="EmployeeId" delimitedColumnName="true" />

<columnOverride column="EmployeeName" delimitedColumnName="true" />

</table>

**参考资料**

本节收集了一些和MyBatis Generator有关的有用的技术主题。

* [从源码构建](http://mbg.cndocs.tk/reference/building.html)
* [扩展 MyBatis Generator](http://mbg.cndocs.tk/reference/extending.html)
* [执行插件](http://mbg.cndocs.tk/reference/pluggingIn.html)
* [日志信息](http://mbg.cndocs.tk/reference/logging.html)

**从源码构建**

所有 MyBatis 代码生成器(MBG)发行版都包含源代码。 运行时的唯一依赖是 ant.jar - 为了圆满完成包含的Ant任务。 它是直接从源代码编译MBG - 只需要解压发行包并使用您喜欢的工具进行编译。

MBG发行包不包含编译期间的测试，或者编译期间必须部分的其他的类。 如果您想检查这些类，或者使用github上最新版本的源码编译MBG，那么按照下面的步骤操作：

1. MBG用Apache Maven构建。 首先您必须能运行maven。 如果您不熟悉Maven，这里有最简单的步骤 (针对 Windows):
   1. 从这里 <http://maven.apache.org/> 下载Maven发行包。
   2. 解压发行包到一个方便的位置。
   3. 设置环境变量和PATH。示例:  
      set JAVA\_HOME=C:\JavaTools\jdk1.6.0\_17  
      set M2\_HOME=C:\JavaTools\apache-maven-3.0  
      set PATH=%PATH%;%M2\_HOME%\bin;
2. 使用git从下面的位置检出代码 <https://github.com/mybatis/generator>
3. 在检出的目录下打开CMD执行mvn clean install命令，或者其他的Maven命令。

**扩展MyBatis Generator**

MyBatis Generator（MBG）是专为扩展性设计的。所有的代码生成使用Java和XML元素简单的DOM表示执行。

Java的DOM包含在org.mybatis.generator.api.dom.java包

该XML DOM包含在org.mybatis.generator.api.dom.xml包

这些类不足以实现每个可以想象到的代码生成的可能性，但是用于生成简单到中等复杂Java和XML代码相当有用的。

在配置文件中使用的选项，您可以提供自己的许多关键代码生成接口的实现。您也可以继承任何提供的实现来提供定制的行为。这个页面将介绍可用的公共API，并提供进一步调查源代码的指示。如果您对如何扩展MBG有任何的困难，您可以在支持的邮件列表[mybatis-user@googlegroups.com](mailto:mybatis-user@googlegroups.com) 发送信息。

**扩展与插入**

虽然本网页上显示了许多不同的扩展点，在大多数情况下，这将是很容易通过使用插件扩展MBG。查阅[开发插件](http://mbg.cndocs.tk/reference/pluggingIn.html)的参考页面了解更多信息。

代码生成的主要扩展点是org.mybatis.generator.api.IntrospectedTable 。实现一个代码生成器是一个有意义的任务，只有当您想要完全取代MBG的代码生成活动时才考虑。自从Abator的原始版本以来，很少出现有增强请求无法通过一个插件来处理的。

**扩展点**

MBG提供了许多不同的扩展点。以下部分列出扩展MBG的不同方法，并描述了可以实现与不同的扩展活动的类型。如果您需要一些了解不同的选择的帮助，您可以自由的在用户邮件列表上询问问题。

**org.mybatis.generator.api.IntrospectedTable**

IntrospectedTable是一个可以扩展用于提供和MBG供给的版本不同的代码生成的抽象类。这种实现的一个很好的例子是基于一个FreeMarker的或Velocity模板的实现。在大多数其他情况下，编写插件是一个更好途径。

如果您选择扩展这个类，您必须提供代码来生成Java和XML文件。您可以选择生成的这些文件的技术。IntrospectedTable基础类持有的一个org.mybatis.generator.internal.rules.Rules实例可查询许多用于代码生成的规则。

MBG提供反射表的几种实现。实现基于<context>元素的targetRuntime属性值。在大多数情况下，继承下面的一个内置的扩展比从头开始创建一个实现会容易很多。下表显示了内置的实现：

|  |  |
| --- | --- |
| **TargetRuntime** | **实现** |
| MyBatis3（默认） | org.mybatis.generator.codegen.mybatis3.IntrospectedTableMyBatis3Impl |
| Ibatis2Java2 | org.mybatis.generator.codegen.ibatis2.IntrospectedTableIbatis2Java2Impl |
| Ibatis2Java5 | org.mybatis.generator.codegen.ibatis2.IntrospectedTableIbatis2Java5Impl |

如果您选择实现此扩展点，指定<context>元素的targetRuntime属性为您的实现类的完全限定类名。

**org.mybatis.generator.api.IntrospectedColumn**

IntrospectedColumn是一个包含列的信息的类，因为它是一个从数据库返回的元数据类。在某些罕见的情况下，可能需要重写这个类来提供自己的 - 特别是如果您创建了一套新的代码生成器。

如果您选择实施此扩展点，指定<context>元素的introspectedColumnImpl属性为您实现类的完全限定类名。

**org.mybatis.generator.api.JavaTypeResolver**

MBG在内省数据库时调用这个接口的方法将JDBC类型映射到Java类型。这个接口的默认实现org.mybatis.generator.internal.types.JavaTypeResolverDefaultImpl 。您可以提供自己的实现，并且默认的实现已被设计为可扩展。

提供您自己的实现，在XML配置中指定全限定的类名：

<javaTypeResolver type="mypackage.MyImplementation“> ...</ javaTypeResolver>

**org.mybatis.generator.api.ShellCallback**

MBG调用这个接口的方法来执行功能，它不能自己做。最重要的功能是：

* 翻译 项​​目/包 到一个目录结构
* 合并事件中的Java源文件同名的现有的Java文件/程序包是否存在。

这个接口的默认实现org.mybatis.generator.internal.DefaultShellCallback 。默认实现简单的将项目和包连接在一起，如果需要则创建必要的包目录。默认的实现不支持Java文件的合并，并将覆盖或忽略文件。

您可以提供自己的实现。如果您想将MBG集成到一些其他的环境中，这将是最重要的一个需要编写的类。例如，Eclipse插件提供的实现在Eclipse环境中运行时，它支持Java文件的合并。

提供您自己的实现，提供有关构造函数的接口org.mybatis.generator.api.MyBatisGenerator的实例对象。这个不能通过XML来配置。如果您提供自己的实现此接口的话，我们假设您也提供了一些附加的代码（如一个新的Ant任务）来运行您的实现。

**org.mybatis.generator.api.ProgressCallback**

MBG调用这个接口的方法在文件的生成（长时间运行的进程）中报告进度。这个接口的默认实现org.mybatis.generator.internal.NullProgressCallback它只是忽略所有的进度消息。您可以提供此接口的实现，支持进度通知和取消代码生成。

当集成MBG到其他IDE环境中实现此接口将是重要的。Eclipse的插件提供此接口挂接到Eclipse的进度通知系统的实现。

提供您自己的实现，在org.mybatis.generator.api.MyBatisGenerator.generate()的一个方法调用上提供其中一个接口的实例。这个不能通过XML来配置。同样，我们假设，如果您提供自己的实现此接口，那么您也提供了一些附加代码（如新的Ant任务或IDE集成）来运行您的实现。

**开发插件**

插件可用于修改或添加到由MyBatis Generator 生成的对象。插件必须实现org.mybatis.generator.api.Plugin接口 。插件接​​口包含许多在代码生成过程的不同阶段被调用的方法。任何特定的插件通常不需要实现整个接口。因此，大多数插件应扩展适配器类org.mybatis.generator.api.PluginAdapter 。适配器类提供基本的插件支持，并为大多数的接口方法（类似于Swing适配器类）提供了空操作的方法。

MyBatis Generator 提供了几个插件（都在包org.mybatis.generator.plugins下 ）。所提供的插件展示了不同类型的可以通过插件完成的任务。插件的源代码是可以下载的，也可以在线浏览[这里](http://mybatis.googlecode.com/svn/sub-projects/generator/trunk/core/mybatis-generator-core/src/main/java/org/mybatis/generator/plugins/) 。

**插件的生命周期**

插件有一个生命周期。插件在代码生成过程的初始化期间创建并且在这个过程中的不同阶段被调用。下面的列表显示了插件的基本生命周期：

1. 插件通过默认的构造函数创建
2. setContext方法被调用
3. setProperties方法被调用
4. validate方法被调用。如果该方法返回false ，那么插件中的其他方法都不会再被调用。
5. 对于配置中的每个表：
   1. initialized方法被调用
   2. Java客户端的方法：1,2
      1. clientXXXMethodGenerated(Method, TopLevelClass, IntrospectedTable) - 当Java客户端实现类生成的时候这些方法被调用.
      2. clientXXXMethodGenerated(Method, Interface, IntrospectedTable) -当Java客户端接口生成的时候这些方法被调用。
      3. clientGenerated(Interface, TopLevelClass, IntrospectedTable)方法被调用
   3. 模型方法：1
      1. modelFieldGenerated, modelGetterMethodGenerated, modelSetterMethodGenerated for each field in the class
      2. modelExampleClassGenerated(TopLevelClass, IntrospectedTable)
      3. modelPrimaryKeyClassGenerated(TopLevelClass, IntrospectedTable)
      4. modelBaseRecordClassGenerated(TopLevelClass, IntrospectedTable)
      5. modelRecordWithBLOBsClassGenerated(TopLevelClass, IntrospectedTable)
   4. SQL映射方法：1
      1. sqlMapXXXElementGenerated(XmlElement, IntrospectedTable) - 当生成SQL映射的每个元素的时候这些方法被调用
      2. sqlMapDocumentGenerated(Document, IntrospectedTable)
      3. sqlMapDocument(GeneratedXmlFile, IntrospectedTable)
   5. contextGenerateAdditionalJavaFiles(IntrospectedTable)方法被调用
   6. contextGenerateAdditionalXmlFiles(IntrospectedTable)方法被调用
6. contextGenerateAdditionalJavaFiles()方法被调用
7. contextGenerateAdditionalXmlFiles()方法被调用

**注意事项:**  
1 -这些方法将被包装的代码生成器调用。如果您提供一个自定义的代码生成器，那么这些方法将仅在自定义代码生成调用它们时调用。  
2 -Java客户端的方法只有当配置Java客户端生成器的时候会被调用。

**编写插件**

实现一个插件，最好的办法是扩展org.mybatis.generator.api.PluginAdapter类并只需要覆盖您需要的插件方法。

插件接口方法可用于修改默认生成的代码，或添加其他生成的代码。Examples of things 可以通过插件实现的是：

* 给生成的方法添加自定义注解
* 给生成的类添加其他方法
* 给生成的XML文件添加其他元素
* 生成额外的Java文件
* 生成额外的XML文件

contextXXX方法总是会被调用。其他方法通过包代码生成器调用 - 且仅当表中的一个规则将会生成一个特定元素时。例如，如果该表没有主键， modelPrimaryKeyClassGenerated(TopLevelClass, IntrospectedTable)方法将不会被调用。

方法返回一个boolean可用于绕过生成代码。如果任意这些方法返回false ，则相关的项目不会被添加到生成的代码中。如果配置了多个插件，那么第一个插件从方法返回false将导致MyBatis Generator停止调用所有其他插件的方法。

如果您有一个关于插件的想法，可以自由的在用户列表问有关的问题。我们在这里提供帮助！

**日志信息**

MyBatis Generator（MBG）报告日志记录的几种不同的方式：

* MBG每次运行的时候会记录并且显示警告信息。 这些消息的目的是告诉用户需要注意可能存在的重要事件，也可能不是。 例如文件正在被覆盖，或者非致命的配置错误等等。 无论配置和命令行参数怎么配置，警告总是会显示。
* MBG生成器每次运行的时候可能会或者不会显示进度信息。 这些信息用来告诉用户代码生成的器的进度信息。 这些信息默认不显示，但是可以通过指定-verbose命令行参数进行显示。 或者您通过Ant任务运行MBG，您可以通过设置verbose 属性为 true, 这样Ant就会运行在verbose模式。
* 最后，MBG将生成跟踪(记录)详细的调试消息。本页说明如何启用这些语句。

一般来说，MBG不会重复消息。所以MBG生成警告的时候这个警告通常还没记录。 在某些情况下需要启用日志记录以及要求MBG记录详细进度消息的时候可能会有用。 这可能会产生大量的输出，但是仍然会提供一个MBG运行期间完整的内部事件描述。

如果Log4J在运行时的类路径中，MBG就会使用Apache Log4J进行记录。 查阅<http://logging.apache.org/log4j/> 获取有关Log4J的详细信息。如果Log4J不在运行时的类路径中，MBG会使用标准的Java日志记录。

即使运行时类路径中存在Log4J，您仍然想强制使用标准的Java日志记录， 您可以在命令行参数指定-forceJavaLogging ，或者当您通过Java执行MBG时使用执行下面的方法：

org.mybatis.generator.logging.LogFactory.forceJavaLogging();

**重要:** 您必须将上面的方法放在任何MBG代码*之前*。

**提供备用的实现**

如果您喜欢使用不同的日志实现比Log4J的或标准的Java日志记录，您可能会提供一个备用实施的关键日志界面如下：

1. 创建org.mybatis.generator.logging.Log接口的实现，实现您选择的日志实现的关键日志方法。
2. 创建org.mybatis.generator.logging.AbstractLogFactory接口的实现，将返回您Log实现的实例。
3. 配置MBG通过调用方法来使用新的LogFactory org.mybatis.generator.logging.LogFactory.setLogFactory(AbstractLogFactory)并提供您的实例AbstractLogFactory实现。

**配置Log4J日志**

下面是一个log4j配置文件示例：

# Set root logger

log4j.rootLogger=INFO, A1

# A1 is set to be a ConsoleAppender.

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-4r %-5p %c - %m%n

# MBG logging configuration...

log4j.logger.org.mybatis.generator=DEBUG

这个文件指示Log4J将MBG所有调试信息输出到控制台。要使用此文件：

1. 运行时类路径的根目录创建一个名为log4j.properties的文件
2. 将上面的条目复制到新文件
3. 运行MBG时，Log4J的JAR文件也需要在运行时类路径下。

您应该可以在控制台中看到很多日志信息。

如果您喜欢，您也可以配置log4j任何其他支持的方法。

**配置Java日志记录**

下面是一个Java日志记录配置的示例文件:

# 指定root logger要创建的处理程序

# (所有的loggers都是root logger的孩子)

handlers = java.util.logging.ConsoleHandler

# 为 root logger 设置默认日志记录级别

.level = INFO

# Set the default logging level for new ConsoleHandler instances

java.util.logging.ConsoleHandler.level = ALL

# Set the default formatter for new ConsoleHandler instances

java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Set the default logging level for the logger named org.mybatis.generator

org.mybatis.generator.level = FINE

这个文件将指导Java写的所有MBG调试消息到控制台。要使用此文件：

1. 创建一个名为logging.properties （或者您喜欢的任何文件名 ​​）的文件。该文件可以在文件系统中存在的任何地方（例如，在一个\temp目录）。
2. 将上述条目复制到新文件
3. 运行MBG时带上这个VM参数：  
   -Djava.util.logging.config.file=\temp\logging.properties （指定您使用的实际的文件名 ​​和目录）

在控制台中您将看到许多日志信息。

如果您喜欢，您还可以配置Java日志记录为其他支持的方法。

**提供的插件**

随着MyBatis生成器（MBG）的使用量的增加，我们发现越来越有用以通过插件增加功能，而不是添加到基本代码发生器的复杂性。插件是一个模块化的，易于理解的机制，继承MBG。有关编写一个插件的更多信息，请参阅[实现插件](http://mbg.cndocs.tk/reference/pluggingIn.html) 。有关配置插件的信息，请参阅[<plugin>](http://mbg.cndocs.tk/configreference/plugin.html)

所提供的插件都在org.mybatis.generator.plugins 包。所提供的插件展示不同类型的可以完成与MBG插件任务。插件的源代码可以和MBG一起下载，也可以在线浏览[这里](https://github.com/mybatis/generator/tree/master/core/mybatis-generator-core/src/main/java/org/mybatis/generator/plugins) 。

**org.mybatis.generator.plugins.CachePlugin**

这个插件在生成的SQL映射中增加了一个<cache>元素。这个插件仅用于MyBatis3目标运行时环境。

这个插件接受下列属性。都是可选的，并且，如果指定，则值将被直接传递到相应的属性生成的<cache>元素。

* cache\_eviction
* cache\_flushInterval
* cache\_readOnly
* cache\_size
* cache\_type

所有属性都可以通过指定<table>元素的属性来覆盖。

**org.mybatis.generator.plugins.CaseInsensitiveLikePlugin**

该插件给Example类添加方法（实际上是给Criteria内部类）来支持不区分大小写的LIKE搜索。这表明通过插件给Example类添加功能，而不是扩展这个类。

**org.mybatis.generator.plugins.EqualsHashCodePlugin**

这个插件给由MBG生成的Java模型对象增加了equals和hashCode方法。

通过这个类生成的equals方法，在大多数情况下是正确的，但如果您已经指定了rootClass可能是不正确的 - 因为我们的equals方法只检查它知道的字段。

**org.mybatis.generator.plugins.MapperConfigPlugin**

这个插件生成包含对MBG所生成的XML映射文件的框架MapperConfig.xml文件。此文件可用于配置的MyBatis 3.X环境。

这个插件接受三个属性：

* fileName ​​（可选的）生成的文件的名称。如果没有指定，默认为“MapperConfig.xml”。
* targetPackage（必须的）生成文件放置的包名。指定像“com.mycompany.sql”这样的值。
* targetProject（必须的）该文件应放置在的项目名称。

注： targetPackage和targetProject遵循sqlMapGenerator配置元素上targetPackage和targetProject的相同的规则。

**org.mybatis.generator.plugins.RenameExampleClassPlugin**

这个插件通过重命名由MBG生成的Example类的方法演示initialized方法的用法。

这个插件接受两个属性：

* searchString（必须的），用于搜索默认生成的Example的名称的正则表达式。
* replaceString（必须的）插在匹配searchString位置的字符串。

例如，从xxxExample重命名生成的例子类xxxCriteria，指定实例美元searchString和标准replaceString

**org.mybatis.generator.plugins.RowBoundsPlugin**

这个插件将添加一个新版本selectByExample方法接受RowBounds参数。这支持的MyBatis RowBounds函数，其中一个返回的结果列表可以在长度受到限制，并且开始位置可以被指定。这可以是在分页应用中是有用的。

这个插件仅适用于MyBatis3目标运行时环境。

**org.mybatis.generator.plugins.SerializablePlugin**

这个插件给由MBG生成的Javas添加了java.io.Serializable标记接口。这个插件给实体类增加了serialVersionUID字段。

重要提示：这是一个简单的实现java.io.Serializable并且不会尝试做任何版本的类。

这个插件接受两个属性：

* addGWTInterface（可选的）True/False.。如果为true，插件将给实体对象增加谷歌Web工具包（GWT）的 IsSerializable接口。默认值是false。
* suppressJavaInterface（必须的）True/False。如果为true，插件将不添加java.io.Serializable接口。这是对于其中对象应该是可序列化的GWT，但不是严格意义上的Java的场景。默认值是false。

**org.mybatis.generator.plugins.SqlMapConfigPlugin**

这个插件生成包含对MBG所产生的SqlMap.xml文件的SqlMapConfig.xml框架文件。此文件可用于配置的iBATIS 2.x的环境。

这个插件接受三个属性：

* fileName ​​（可选的）所产生的文件的名称。如果没有指定，默认值是“SqlMapConfig.xml”。
* targetPackage（必须的）放置该文件的包名。指定像“com.mycompany.sql”这样的值。
* targetProject（必须的）放置该文件的项目的名称。

注： targetPackage和targetProject遵循和sqlMapGenerator配置元素上的targetPackage和targetProject相同的规则。

**org.mybatis.generator.plugins.ToStringPlugin**

该插件给实体类添加toString()方法。

**org.mybatis.generator.plugins.VirtualPrimaryKeyPlugin**

这个插件可用于指定作为主键的列，即使它们没有在数据库中被定义为主键列。这是在数据库表没有定义主的情况下非常有用。通常情况下，如果没有主键，MBG将产生一组非常有限的方法。这个插件可以用来启用生成的完整的MBG方法。

要使用该插件，添加属性“virtualKeyColumns”到您的<table>配置，设置值为应被视做主键的用逗号或空格分隔的列名列表。列名必须和数据库（通常全部大写）返回的列名完全匹配。例如：

<table tableName="foo">

<property name="virtualKeyColumns" value="ID1, ID2" />

</table>

**设计理念**

此工具可能会引发一些哲理性问题， 因为该工具是更侧重于数据库表，而不是实体类。 我们将采取几个段落来谈论这种做法。

首先，这个工具做什么。我们不是在做任何有关项目应该或不应该是结构化这种说法的。 在一般情况下我们支持富实体类， 但创建一个富实体类和是否应该坚持这种模式完全不是一码事。

如果您特定的设计理念是实体类驱动所有的决策，数据库设计是屈从于实体类， 那么这个工具 - 和MyBatis本身 - 可能不适合您的应用程序。 在这种情况下，我们建议您认真考虑 [Hibernate](http://www.hibernate.org) - 他可能更贴近您的应用程序和设计理念。

但并非所有的项目都适合这种模式。很少有真正的企业级应用程序会这么做。 MyBatis对那些将数据库表设计和实体对象设计看做一样的项目有很大的帮助。 MyBatis不是对象关系映射，并且不会尝试去持久化对象。所以他着眼于应用开发人员编写SQL和数据库表进行交互。

在大型或企业级的项目，其中的许多因素是很常见:

* 数据库设计往往是从 OO 实体设计一个单独的功能（有单独的管理）
* 数据库设计者没有 OO 工具 （比如继承），所以他们不从 OO 考虑。
* 应用程序设计人员并不能完全控制数据库表的最后形式。 例如，在应用中适合存在于一个对象的数据，在数据库中可能被分成几个表。
* 数据库设计最终和OO设计的完全不同，从而导致表和对象之间的明显不能匹配。

从这些主要的指标来看，对您的应用来说 MyBatis 是一个非常不错的候选工具，MyBatis Generator 可以在这类项目起到重要的影响。 那么，在这种情况下应该如何使用MyBatis?

数据访问对象(DAO)模式仍然是一种主要的模式。MyBatis Generator可以生成一组基本的和每个表对应的对象。 生成的代码是和事务无关的。这意味着如果在一个事务中涉及多个表参与，可以很容易的扩展生成的代码。 或者您可以创建另一个DAO（或服务方法），更紧密地匹配实体对象的持久性需求，并可以再一个事务中使用一个或多个生成的对象。

举个例子，考虑一个典型的 Order 对象，典型的标题和细节的问题。 在某些环境中这种对象将保存到至少四个表 - 两个关键表、"标题"表和"详细信息"表 (再次声明，我们不做任何种类的关于这是否"正确"的语句设计，只在陈述一个事实)。 您的应用程序还应与 Order 对象进行交互, 并且可能在某个地方（在OrderDAO 或者在一个Service对象）存在一个saveOrder(Order order)方法 该方法将与所涉及的4个表生成的代码进行交互。

在这种情况下，代码生成器给我们带来什么? 有以下几种:

* 重用（复用） - 很可能是一些表将需要从不同的DAO或者Service方法访问。 为每个表创建一个 DAO 可以促进重用和应用程序内的一致性。
* 数据库抽象 - 服务层通常在应用程序中定义持久性。这些方法可以很快稳定。 随着数据库设计的发展：
  1. 代码可以很快的随着表的改变而重新生成
  2. Service方法可以根据需要修改
  3. 更高层的应用程序保持不变
* 开发者的效率 - 生成基于表的DAO是很快速、可重复和无差错的。 开发人员可以专注于对象持久化和可能需要的复杂的联接查询。
* 更少的缺陷 - 因为所有应用程序中最繁琐和最容易出错的部分（获得SQL匹配的对象）是自动化的。

# [Java Persistence with MyBatis 3(中文版)](http://blog.csdn.net/luanlouis/article/details/35567875)

# Mybatis Generator最完整配置详解