# Maven插件

1. <https://www.yiibai.com/maven/maven_plugins.html>

2. <https://www.iteye.com/blog/juvenshun-232916>

## 一、什么是Maven的插件?

Maven是一个执行插件的框架，每一个任务实际上是由插件完成的。

每个生命周期中都包含着一系列的阶段（Phase），这些阶段就相当于Maven提供的统一的接口，然后这些phase的实现是由Maven的插件来完成。

如果你理解设计模式中的模板方法（Template Method）模式，那么这会变得很好理解。

在模板方法模式中，父类定义了一些抽象方法，并且决定这些方法的调用顺序，子类只需要实现这些方法。

比如沏茶这个用例，父类定义的抽象方法有：烧开水，洗茶杯，倒水，放茶叶，并且有一个沏茶方法来顺序调用这些方法。而子类会去实现如何烧开水，如何洗茶杯，如何倒水，如何放茶叶。实例化一个子类之后，调用沏茶方法，这些步骤就会依次执行了。

同样的道理，在Maven中，生命周期定义了编译，测试，打包，部署等等阶段，而插件会去实现这些对应的阶段，如上面提到的Compiler插件实现了如何编译。

一个插件通常提供了一组目标，而目标通常是和生命周期中的某个阶段绑定在一起，一个插件目标可以绑定到多个生命周期的阶段，可使用如下语法来执行

mvn [plugin-name]:[goal-name]

## 二、插件类型：

Maven提供了以下两种类型插件：

|  |  |
| --- | --- |
| 类型 | 描述 |
| 构建插件 | 在生成过程中执行,并在pom.xml中的<build/>元素进行配置 |
| 报告插件 | 在网站生成期间执行,在pom.xml中的<reporting/>元素进行配置 |

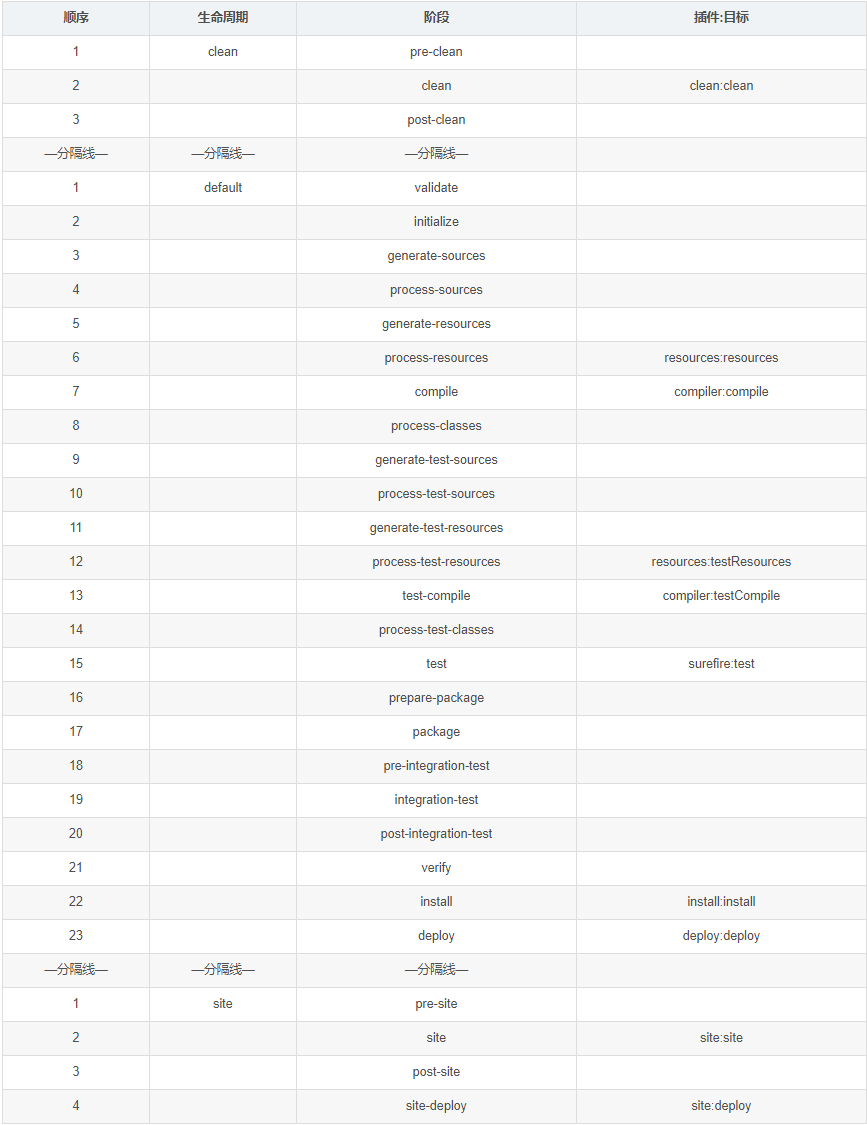
## 三、常见插件列表：

|  |  |
| --- | --- |
| 插件 | 描述 |
| clean | 编译后的清理目标,删除目标目录 |
| compile | 编译Java源文件 |
| surefire | 进行单元测试,创建测试报告 |
| jar | 从当前项目构建JAR文件 |
| war | 从当前项目构建WAR文件 |
| javadoc | 产生用于该项目的javadoc |
| antrun | 从构建的任何阶段运行一组Ant任务 |

## 四、默认生命周期阶段与插件目标的绑定关系：

Maven为实现快速构建，有一套内置的插件绑定。

三套生命周期的插件绑定如下（其实是各个生命周期阶段与插件目标的绑定）



## 五、示例：

### 1、antrun插件：

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.clean</id>

<phase>clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>clean phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

【解读】

这段代码的意思是：

当进行到了Clean生命周期的clean阶段(<phase>clean</phase>)的时候，执行antrun插件(maven-antrun-plugin)的run目标(<goal>run</goal>)，而该目标所要做的任务是输出“clean phase”(<tasks><echo>clean phase</echo></tasks>)。

### 2、Tomcat插件：

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-catalina -->

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.1</version>

<configuration>

<path>/</path>

<port>8080</port>

</configuration>

</plugin>

</plugins>

</build>

</project>

配置了Tomcat插件之后，运行web项目的时候，就不用我们本地下载Tomcat服务器

### 3、compile插件：

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<!-- 配置编译插件 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<encoding>utf-8</encoding>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin></plugins>

</build>

</project>

### 4、help插件：

Maven Help插件有四个目标：

1. help:active-profiles

列出当前构建中活动的profile（项目的，用户的，全局的）

1. help:effective-pom

显示当前构建的实际POM，包含活动的profile

1. help:effective-settings

打印出项目的实际settings，包括从全局的settings和用户的settings继承的配置。

1. help:describe

描述插件的属性，它不需要在项目目录下运行。但是你必须提供你想要描述插件的groupId和artifactId。

### source plugin：

内置的插件绑定关系并没有涉及创建项目的源码包这一任务，但我们可以自己配置source插件来完成。

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-source-plugin</artifactId>  <version>2.1.1</version>  <executions>  <execution>  <id>attach-sources</id>  <phase>verify</phase>  <goals>  <goal>jar-no-fork</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

## 六、描述一个Maven插件：

一旦你开始使用Maven，你会花很多时间去试图获得Maven插件的信息：插件如何工作？配置参数是什么？目标是什么？你会经常使用help:describe目标来获取这些信息。通过plugin参数，你可以指定想要研究哪个插件，你可以传入插件的前缀，如help就是maven-help-plugin插件的前缀，或者可以是groupId:artifact[:version]，这里的version是可选的。比如，下面的命令使用help插件的describe目标来输出Maven Help插件的信息。

C:\Dev\Repository\ProjectRepository\HelloProject>mvn help:describe -Dplugin=help

[INFO] Scanning for projects...

[INFO]

[INFO] ---------------------< com.chenming:HelloProject >----------------------

[INFO] Building HelloProject 1.0

[INFO] --------------------------------[ pom ]---------------------------------

[INFO]

[INFO] --- maven-help-plugin:3.2.0:describe (default-cli) @ HelloProject ---

[INFO] org.apache.maven.plugins:maven-help-plugin:3.2.0

Name: Apache Maven Help Plugin

Description: The Maven Help plugin provides goals aimed at helping to make

sense out of the build environment. It includes the ability to view the

effective POM and settings files, after inheritance and active profiles have

been applied, as well as a describe a particular plugin goal to give usage

information.

Group Id: org.apache.maven.plugins

Artifact Id: maven-help-plugin

Version: 3.2.0

Goal Prefix: help

This plugin has 8 goals:

help:active-profiles

Description: Displays a list of the profiles which are currently active for

this build.

help:all-profiles

Description: Displays a list of available profiles under the current

project.

Note: it will list all profiles for a project. If a profile comes up with a

status inactive then there might be a need to set profile activation

switches/property.

help:describe

Description: Displays a list of the attributes for a Maven Plugin and/or

goals (aka Mojo - Maven plain Old Java Object).

help:effective-pom

Description: Displays the effective POM as an XML for this build, with the

active profiles factored in, or a specified artifact. If verbose, a comment

is added to each XML element describing the origin of the line.

help:effective-settings

Description: Displays the calculated settings as XML for this project,

given any profile enhancement and the inheritance of the global settings

into the user-level settings.

help:evaluate

Description: Evaluates Maven expressions given by the user in an

interactive mode.

help:help

Description: Display help information on maven-help-plugin.

Call mvn help:help -Ddetail=true -Dgoal=<goal-name> to display parameter

details.

help:system

Description: Displays a list of the platform details like system properties

and environment variables.

For more information, run 'mvn help:describe [...] -Ddetail'

通过设置plugin参数来运行describe目标，输出为该插件的Maven坐标，目标前缀，和该插件的一个简要介绍。尽管这些信息非常有帮助，你通常还是需要了解更多的详情。如果你想help插件输出完整的带有参数的目标列表，只要运行带有参数detail的help:describe目标就可以了，像这样：

C:\Dev\Repository\ProjectRepository\HelloProject>mvn help:describe -Dplugin=help -Ddetail

[INFO] Scanning for projects...

[INFO]

[INFO] ---------------------< com.chenming:HelloProject >----------------------

[INFO] Building HelloProject 1.0

[INFO] --------------------------------[ pom ]---------------------------------

[INFO]

[INFO] --- maven-help-plugin:3.2.0:describe (default-cli) @ HelloProject ---

[INFO] org.apache.maven.plugins:maven-help-plugin:3.2.0

Name: Apache Maven Help Plugin

Description: The Maven Help plugin provides goals aimed at helping to make

sense out of the build environment. It includes the ability to view the

effective POM and settings files, after inheritance and active profiles have

been applied, as well as a describe a particular plugin goal to give usage

information.

Group Id: org.apache.maven.plugins

Artifact Id: maven-help-plugin

Version: 3.2.0

Goal Prefix: help

This plugin has 8 goals:

help:active-profiles

Description: Displays a list of the profiles which are currently active for

this build.

Implementation: org.apache.maven.plugins.help.ActiveProfilesMojo

Language: java

Available parameters:

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

help:all-profiles

Description: Displays a list of available profiles under the current

project.

Note: it will list all profiles for a project. If a profile comes up with a

status inactive then there might be a need to set profile activation

switches/property.

Implementation: org.apache.maven.plugins.help.AllProfilesMojo

Language: java

Available parameters:

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

help:describe

Description: Displays a list of the attributes for a Maven Plugin and/or

goals (aka Mojo - Maven plain Old Java Object).

Implementation: org.apache.maven.plugins.help.DescribeMojo

Language: java

Available parameters:

artifactId

User property: artifactId

The Maven Plugin artifactId to describe.

Note: Should be used with groupId parameter.

cmd

User property: cmd

A Maven command like a single goal or a single phase following the Maven

command line:

mvn [options] [<goal(s)>] [<phase(s)>]

detail (Default: false)

User property: detail

This flag specifies that a detailed (verbose) list of goal (Mojo)

information should be given.

goal

User property: goal

The goal name of a Mojo to describe within the specified Maven Plugin. If

this parameter is specified, only the corresponding goal (Mojo) will be

described, rather than the whole Plugin.

groupId

User property: groupId

The Maven Plugin groupId to describe.

Note: Should be used with artifactId parameter.

minimal (Default: false)

User property: minimal

This flag specifies that a minimal list of goal (Mojo) information should

be given.

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

plugin

Alias: prefix

User property: plugin

The Maven Plugin to describe. This must be specified in one of three

ways:

1. plugin-prefix, i.e. 'help'

2. groupId:artifactId, i.e. 'org.apache.maven.plugins:maven-help-plugin'

3. groupId:artifactId:version, i.e.

'org.apache.maven.plugins:maven-help-plugin:2.0'

version

User property: version

The Maven Plugin version to describe.

Note: Should be used with groupId/artifactId parameters.

help:effective-pom

Description: Displays the effective POM as an XML for this build, with the

active profiles factored in, or a specified artifact. If verbose, a comment

is added to each XML element describing the origin of the line.

Implementation: org.apache.maven.plugins.help.EffectivePomMojo

Language: java

Available parameters:

artifact

User property: artifact

The artifact for which to display the effective POM.

Note: Should respect the Maven format, i.e. groupId:artifactId[:version].

The latest version of the artifact will be used when no version is

specified.

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

verbose (Default: false)

User property: verbose

Output POM input location as comments.

help:effective-settings

Description: Displays the calculated settings as XML for this project,

given any profile enhancement and the inheritance of the global settings

into the user-level settings.

Implementation: org.apache.maven.plugins.help.EffectiveSettingsMojo

Language: java

Available parameters:

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

showPasswords (Default: false)

User property: showPasswords

For security reasons, all passwords are hidden by default. Set this to

true to show all passwords.

help:evaluate

Description: Evaluates Maven expressions given by the user in an

interactive mode.

Implementation: org.apache.maven.plugins.help.EvaluateMojo

Language: java

Available parameters:

artifact

User property: artifact

An artifact for evaluating Maven expressions.

Note: Should respect the Maven format, i.e. groupId:artifactId[:version].

The latest version of the artifact will be used when no version is

specified.

expression

User property: expression

An expression to evaluate instead of prompting. Note that this must not

include the surrounding ${...}.

forceStdout (Default: false)

User property: forceStdout

This options gives the option to output information in cases where the

output has been suppressed by using -q (quiet option) in Maven. This is

useful if you like to use maven-help-plugin:evaluate in a script call

(for example in bash) like this:

RESULT=$(mvn help:evaluate -Dexpression=project.version -q -DforceStdout)

echo $RESULT

This will only printout the information which has been requested by

expression to stdout.

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console. This parameter will be ignored if no

expression is specified.

Note: Could be a relative path.

help:help

Description: Display help information on maven-help-plugin.

Call mvn help:help -Ddetail=true -Dgoal=<goal-name> to display parameter

details.

Implementation: org.apache.maven.plugins.help.HelpMojo

Language: java

Available parameters:

detail (Default: false)

User property: detail

If true, display all settable properties for each goal.

goal

User property: goal

The name of the goal for which to show help. If unspecified, all goals

will be displayed.

indentSize (Default: 2)

User property: indentSize

The number of spaces per indentation level, should be positive.

lineLength (Default: 80)

User property: lineLength

The maximum length of a display line, should be positive.

help:system

Description: Displays a list of the platform details like system properties

and environment variables.

Implementation: org.apache.maven.plugins.help.SystemMojo

Language: java

Available parameters:

output

User property: output

Optional parameter to write the output of this help in a given file,

instead of writing to the console.

Note: Could be a relative path.

## 自定义绑定：

除了内置绑定之外，用户还能自己选择将某个插件目标绑定到生命周期的某个阶段上，这种自定义绑定方式能让Maven项目在构建过程中执行更多更富特色的任务。

一个常见的例子是创建项目的源码jar包，内置的插件绑定关系中并没有涉及这一任务，因此需要用户自行配置。maven-source-plugin可以帮助我们完成该任务，它的jar-no-fork目标能够将项目的主代码打包成jar包，可以将其绑定到default生命周期的verify阶段上，在执行完成集成测试后和安装构件之前创建源码jar包。具体配置见代码清单7-3：

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-source-plugin</artifactId>  <version>2.1.1</version>  <executions>  <execution>  <id>attach-sources</id>  <phase>verify</phase>  <goals>  <goal>jar-no-fork</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

在POM的build元素下的plugins子元素中声明插件的使用，该例中用到的是maven-source-pugin，其groupId为org.apache.maven.plugins，这也是Maven官方插件的groupId，紧接着的artifactId为maven-source-plugin，version是2.1.1。对于自定义绑定的插件，用户总是应该声明一个非快照版本，这样可以避免由于插件版本变化造成的构件不稳定性。

上诉配置中，除了基本的插件坐标声明外，还有插件执行配置，executions下每个execution子元素可以用来配置执行一个任务。该例中配置了一个id为attach-sources（id随意，只要唯一就行）的任务，通过phase配置，将其绑定到Default生命周期的verify阶段上，在通过goals配置指定要指定的插件目标。至此，自定义插件绑定完成。

有时候，即使不通过phase元素配置声明周期阶段，插件目标也能够绑定到生命周期中去。例如，可以尝试删除上述配置中的phase一行，再次执行mvn verify，仍然可以看到maven-source-plugin:jar-no-fork得以执行。出现这种现象的原因是：有很多插件的目标在编写时已经定义了默认绑定阶段。可以使用maven-help-plugin查看插件详细信息。

我们知道，当插件目标被绑定到不同生命周期阶段的时候，其执行顺序会由生命周期阶段的先后顺序决定。如果多个目标被绑定到同一个阶段，它们的执行顺序会是怎样？答案很简单，当多个插件目标绑定到同一个阶段的时候，这些插件声明的顺序决定了目标的执行顺序。

## 插件解析机制：

为了方便用户使用和配置插件，Maven不需要用户提供完整的插件坐标信息，就可以解析得到正确的插件，Maven的这一特性是一把双刃剑，虽然它简化了插件的使用和配置，可一旦插件的行为出现异常，用户就很难快速定位到出问题的插件构件。例如mvn help:system这样一条命令，它到底执行了什么插件？该插件的groupId、artifactId和version分别是什么？这个构件是从哪里来的？本届就详细介绍Maven的运行机制，以让读者不仅知其然，更知其所以然。

### 插件仓库：

与依赖构件一样，插件构件同样基于坐标存储在Maven仓库中。在需要的时候，Maven会从本地仓库寻找插件，如果不存在，则从远程插件仓库查找。找到插件之后，再下载到本地仓库使用。

值得一提的是，Maven会区别对待依赖的远程仓库与插件的远程仓库，之前介绍了如何配置远程仓库，但那种配置只对一般依赖有效果。当Maven需要的依赖在本地仓库不存在时，它会去所配置的远程仓库中查找，可当Maven需要的插件在本地仓库存在时，它就不会去这些远程仓库查找。

不同于repositories及其repository子元素，插件的远程仓库使用pluginRepositories和pluginRepository配置。例如，Maven内置了如下的插件远程仓库配置，见代码清单7-7：

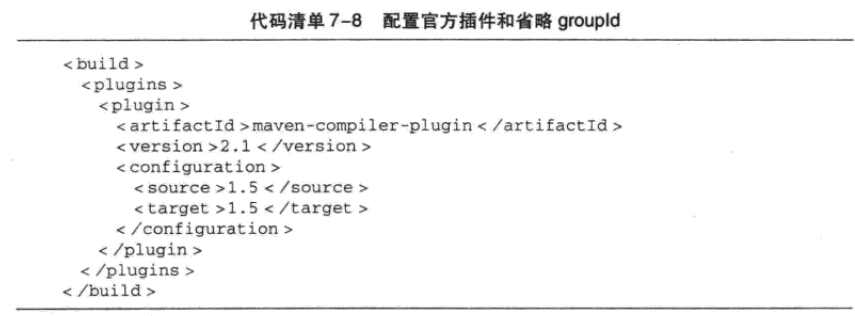


除了pluginRepositories和pluginRepository标签不同之外，其余所有子元素表达的含义与第6.4节所介绍的依赖远程仓库配置完全一样。我们甚至看到，这个默认插件仓库的地址就是中央仓库，它关闭了对SNAPSHOT的支持，以防止引入SNAPSHOT版本的插件而导致不稳定的构建。

一般来说，中央仓库所包含的插件完全能够满足我们的需要，因为也不需要配置其他的插件仓库。只有在很少的情况下，项目使用的插件无法在中央仓库找到，或者自己编写了插件，这个时候可以参考上述的配置，在POM或者settings.xml中加入其他的插件仓库配置。

### 插件的默认groupId：

在POM中配置插件的时候，如果该插件是Maven的官方插件（即如果groupId为org.apache.maven.plugins），就可以省略groupId配置，见代码清单7-8：

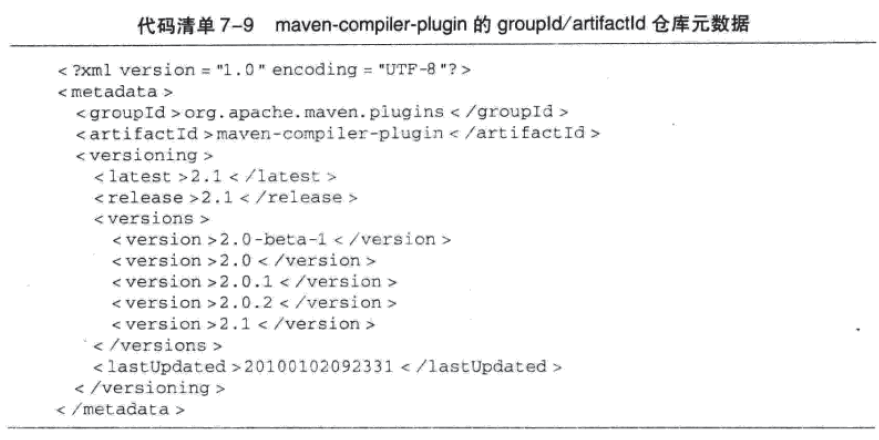


### 解析插件版本：

同样是为了简化插件的配置和使用，在用户没有提供插件版本的情况下，Maven会自动解析插件版本。

首先，Maven在超级POIM中为所有核心插件设定了版本，超级POM是所有Maven项目的父POM，所有项目都继承这个超级POM的配置，因此，即使用户不加任何配置，Maven使用核心插件的时候，它们的版本就已经确定了。这些插件包括maven-clean-plugin、maven-compiler-pugin、maven-surefire-plugin等。

如果用户使用某个插件时没有指定版本，而这个插件又不属于核心插件的范畴，Maven就会去检查所有仓库中可用的版本，然后做出选择。读者可以回顾一下之前介绍的仓库元数据groupId/artifactId/maven-metadata.xml。以maven-compiler-plugin为例，它在中央仓库的仓库元数据为http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/maven-metadata.xml，其内容见代码清单7-9：



Maven遍历本地仓库和所有远程插件仓库，将该路径下的仓库元数据归并后，就能计算出latest和release的值。latest表示所有仓库中该构件的最新版本，而release表示最新的非快照版本。在Maven2中，插件的版本会被解析至latest。也就是说，当用户使用某个非核心插件且没有声明版本的时候，Maven会将版本解析为所有可用仓库中的最新版本，而这个版本也可能是快照版本。

当插件的版本为快照版本的时候，就会出现潜在的问题。Maven会基于更新策略，检查并使用快照的更新。某个插件可能昨天还用的好好的，今天就出错了，其原因就是这个快照版本的插件发生了变化。

为了防止此类问题，Maven3调整了解析机制，当插件没有声明版本的时候，不再解析至latest，而是使用release。这样就可以避免由于快照频繁更新而导致的插件行为不稳定。

依赖Maven解析插件版本是不推荐的做法，即使Maven3将版本解析到最新的非快照版本，也还是会有潜在的不稳定性。例如，可能某个插件发布了一个新的版本，而这个版本的行为与之前的版本发生了变化，这种变化可能导致项目构建失败。因此，使用插件的时候，应该一直显式的设定版本，这也解释了Maven为什么要在超级POM中为核心插件设定版本。

### 解析插件前缀：

前面讲到mvn命令行支持使用插件前缀来简化插件的调用，现在解释Maven如何根据插件前缀解析得到插件的坐标。

插件前缀与groupId:artifactId是一一对应的，这种匹配关系存储在仓库元数据中。与之前提到的groupId/artifactId/maven-metadata.xml不同，这里的仓库元数据为groupId/maven-metadata.xml，那么这里的groupId是什么呢？之前提到主要的插件都位于http://repo1.maven.org/maven2/org/apache/maven/plugins和http://repository.codehuas.org/org/codehuas/mojo/，相应的，Maven在解析插件仓库元数据的时候，会默认使用org.apache.maven.plugins和org.codehuas.mojo这两个groupId。也可以通过配置settings.xml让Maven检查其他groupId上的插件仓库元数据：

|  |
| --- |
| <settings>  <pluginGroups>  <pluginGroup>com.your.plugins</pluginGroup>  </pluginGroups>  <settings> |

基于该配置，Maven就不仅仅会检查org/apache/maven/plugins/maven-metadata.xml和org/codehuas/mojo/maven-metadata.xml，还会检查com/your/plugins/maven-metadata.xml。

下面看一个插件仓库元数据的内容，见代码清单7-10：



当Maven解析到dependency:tree这样的命令后，它首先基于默认的groupId归并所有插件仓库的元数据org/apache/maven/plugins/maven-metadata.xml；其次检查归并后的元数据，找到对应的artifactId为maven-dependency-plugin；然后结合大于元数据的groupId org.apache.maven.plugins；最后使用之前描述的方法解析得到version，这时就得到了完整的插件坐标。如果org/apache/maven/plugins/maven-metadata.xml没有记录该插件前缀，则接着检查其他groupId下的元数据，如org/codehuas/mojo/maven-metadata.xml，以及用户自定义的插件组。如果所有元数据中都不包含该前缀，则报错。

### 配置插件前缀：

除了上面(4)所述的配置插件前缀，还可以直接在pom.xml中直接配置插件前缀：

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-plugin-plugin</artifactId>  <version>3.2</version>  <configuration>  <goalPrefix>hello</goalPrefix>  <skipErrorNoDescriptorsFound>true</skipErrorNoDescriptorsFound>  </configuration>  </plugin>  </plugins> </build> |

此时执行 mvn hello:count 就可以执行Hello-plugin插件的count目标

