### Maven的Java插件开发指南

[原文地址](https://maven.apache.org/guides/plugin/guide-java-plugin-development.html" \t "http://ifeve.com/maven-java-pluging/_blank) 译者：刘小刘

## 介绍

本指南的目标是帮助用户使用Java开发Maven插件。

### 重要声明：插件命名规范和Apache Maven商标

通常将你的插件命名为<yourplugin>-maven-plugin。  
强烈反对命名为maven-<yourplugin>-plugin（maven在插件名开头），因为这是Apache Maven组织为官方Maven插件保留的命名格式，用org.apache.maven.plugins作为组id。使用此命名是对Apache Maven商标的侵权。

### 你的第一个插件

这一部分我们将构建一个不带参数、运行时输出一行信息的简单插件，会覆盖创建插件项目的基本操作、Java Mojo的核心内容，以及执行Mojo的两种方式。

#### 你的第一个Mojo

最简单地，一个Java Mojo只包含一个类。不需要像EJB有多个类，尽管包含一组相似Mojo的插件很可能提取一个抽象基类以合并公共代码。  
在代码树中搜索Mojo时，plugin-tools寻找带有@Mojo形式的Java5注解的类或带有goal的javadoc型注解。所有带有这样注解的类包含在插件配置文件中。

##### 一个简单的Mojo

下面是一个简单的无参数Mojo类，它可能是最简单的Mojo了。代码示例后面是各部分的说明。

|  |  |
| --- | --- |
| 01 | **package** sample.plugin; |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | **import** org.apache.maven.plugin.AbstractMojo; |

|  |  |
| --- | --- |
| 04 | **import** org.apache.maven.plugin.MojoExecutionException; |

|  |  |
| --- | --- |
| 05 | **import** org.apache.maven.plugins.annotations.Mojo; |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | /\*\* |

|  |  |
| --- | --- |
| 08 | \* Says "Hi" to the user. |

|  |  |
| --- | --- |
| 09 | \* |

|  |  |
| --- | --- |
| 10 | \*/ |

|  |  |
| --- | --- |
| 11 | @Mojo( name = "sayhi") |

|  |  |
| --- | --- |
| 12 | **public** **class** GreetingMojo **extends** AbstractMojo |

|  |  |
| --- | --- |
| 13 | { |

|  |  |
| --- | --- |
| 14 | **public** **void** execute() **throws** MojoExecutionException |

|  |  |
| --- | --- |
| 15 | { |

|  |  |
| --- | --- |
| 16 | getLog().info( "Hello, world." ); |

|  |  |
| --- | --- |
| 17 | } |

|  |  |
| --- | --- |
| 18 | } |

* 类org.apache.maven.plugin.AbstractMojo提供了实现一个Mojo需要的大部分基础功能，除了execute方法。
* “@Mojo”注解是必须的，它控制了Mojo何时和怎样被执行
* execute方法可以抛出两种异常：
* org.apache.Maven.plugin.MojoExecutionException 如果发生了非预期的错误抛出此异常，显示”BUILD ERROR”信息。
* org.apache.Maven.plugin.MojoFailureException 如果发生了预期内的错误（例如编译错误）抛出此异常，显示“BUILD FAILURE”信息。
* getLog方法（在AbstractMojo中定义）返回一个类似log4j的日志对象，插件可以用它输出debug, info, warn, error级别的日志。这个日志是用户可接受的显示信息的方式。查看[Retrieving the Mojo Logger](https://maven.apache.org/plugin-developers/common-bugs.html" \l "Retrieving_the_Mojo_Logger) 了解如何恰当使用。

所有的Mojo注解在 [Mojo API Specification](https://maven.apache.org/developers/mojo-api-specification.html" \l "The_Descriptor_and_Annotations)中描述。

#### 项目定义

插件的Mojo写完后，就可以构建插件了。插件描述中需要设置以下几项：

|  |  |
| --- | --- |
| groupId | 插件的组id，应当与Mojo包名的共同前缀一致 |
| artifactId | 插件名 |
| version | 插件版本 |
| packaging | 应设为”maven-plugin” |
| dependencies | 必须声明对Maven Plugin Tools API的依赖以解析”AbstractMojo”和相关类 |

下面是示例Mojo的pom示例，参数按上表要求设定：

|  |  |
| --- | --- |
| 01 | <project> |

|  |  |
| --- | --- |
| 02 | <modelVersion>4.0.0</modelVersion> |

|  |  |
| --- | --- |
| 03 |  |

|  |  |
| --- | --- |
| 04 | <groupId>sample.plugin</groupId> |

|  |  |
| --- | --- |
| 05 | <artifactId>hello-maven-plugin</artifactId> |

|  |  |
| --- | --- |
| 06 | <version>1.0-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 07 | <packaging>maven-plugin</packaging> |

|  |  |
| --- | --- |
| 08 |  |

|  |  |
| --- | --- |
| 09 | <name>Sample Parameter-less Maven Plugin</name> |

|  |  |
| --- | --- |
| 10 |  |

|  |  |
| --- | --- |
| 11 | <dependencies> |

|  |  |
| --- | --- |
| 12 | <dependency> |

|  |  |
| --- | --- |
| 13 | <groupId>org.apache.maven</groupId> |

|  |  |
| --- | --- |
| 14 | <artifactId>maven-plugin-api</artifactId> |

|  |  |
| --- | --- |
| 15 | <version>3.0</version> |

|  |  |
| --- | --- |
| 16 | </dependency> |

|  |  |
| --- | --- |
| 17 |  |

|  |  |
| --- | --- |
| 18 | <!-- dependencies to annotations --> |

|  |  |
| --- | --- |
| 19 | <dependency> |

|  |  |
| --- | --- |
| 20 | <groupId>org.apache.maven.plugin-tools</groupId> |

|  |  |
| --- | --- |
| 21 | <artifactId>maven-plugin-annotations</artifactId> |

|  |  |
| --- | --- |
| 22 | <version>3.4</version> |

|  |  |
| --- | --- |
| 23 | <scope>provided</scope> |

|  |  |
| --- | --- |
| 24 | </dependency> |

|  |  |
| --- | --- |
| 25 | </dependencies> |

|  |  |
| --- | --- |
| 26 | </project> |

#### 构建目标

在maven-plugin中定义了很少几个目标作为标准构建生命周期的一部分：

|  |  |
| --- | --- |
| compile | 编译插件的Java代码，构建插件描述 |
| test | 运行插件的单元测试 |
| package | 构建插件jar包 |
| install | 将插件jar安装到本地仓库 |
| deploy | 将插件jar部署到远程仓库 |

#### 执行你的第一个Mojo

执行新插件的最直接方式是在命令行直接指定插件目标。要这样做，你需要在项目中这样配置hello-Maven-plugin：

|  |  |
| --- | --- |
| 01 | ... |

|  |  |
| --- | --- |
| 02 | <build> |

|  |  |
| --- | --- |
| 03 | <plugins> |

|  |  |
| --- | --- |
| 04 | <plugin> |

|  |  |
| --- | --- |
| 05 | <groupId>sample.plugin</groupId> |

|  |  |
| --- | --- |
| 06 | <artifactId>hello-maven-plugin</artifactId> |

|  |  |
| --- | --- |
| 07 | <version>1.0-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 08 | </plugin> |

|  |  |
| --- | --- |
| 09 | </plugins> |

|  |  |
| --- | --- |
| 10 | </build> |

|  |  |
| --- | --- |
| 11 | ... |

然后，指定目标的全名：

|  |  |
| --- | --- |
| 1 | mvn groupId:artifactId:version:goal |

例如，要运行示例插件的Mojo，在命令行输入“mvn sample.plugin:hello-Maven-plugin:1.0-SNAPSHOT:sayhi”。  
注：运行单一目标不需要指定版本。

##### 缩短命令行

有几种方式缩短输入：

* 如果运行本地仓库安装的最新版插件，可以省略版本号。使用”mvn sample.plugin:hello-Maven-plugin:sayhi”运行。
* 赋予插件一个短前缀，如mvn hello:sayhi。如果按照${prefix}-maven-plugin的命名方式（如果插件是Apache Maven的官方插件用maven-${prefix}-plugin）这是自动完成的。也可以通过额外的配置设置前缀，更多信息参见[Introduction to Plugin Prefix Mapping](https://maven.apache.org/guides/introduction/introduction-to-plugin-prefix-mapping.html)。
* 最后，可以把插件的组id加入默认搜索的组id列表。这种方式需要在${user.home}/.m2/settings.xml中增加如下配置：

|  |  |
| --- | --- |
| 1 | <pluginGroups> |

|  |  |
| --- | --- |
| 2 | <pluginGroup>sample.plugin</pluginGroup> |

|  |  |
| --- | --- |
| 3 | </pluginGroups> |

此时，可以用”mvn hello:sayhi”运行Mojo了。

##### 将Mojo关联到构建生命周期

你也可以配置插件关联到构建生命周期某个特定阶段的指定目标，示例如下：

|  |  |
| --- | --- |
| 01 | <build> |

|  |  |
| --- | --- |
| 02 | <plugins> |

|  |  |
| --- | --- |
| 03 | <plugin> |

|  |  |
| --- | --- |
| 04 | <groupId>sample.plugin</groupId> |

|  |  |
| --- | --- |
| 05 | <artifactId>hello-maven-plugin</artifactId> |

|  |  |
| --- | --- |
| 06 | <version>1.0-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 07 | <executions> |

|  |  |
| --- | --- |
| 08 | <execution> |

|  |  |
| --- | --- |
| 09 | <phase>compile</phase> |

|  |  |
| --- | --- |
| 10 | <goals> |

|  |  |
| --- | --- |
| 11 | <goal>sayhi</goal> |

|  |  |
| --- | --- |
| 12 | </goals> |

|  |  |
| --- | --- |
| 13 | </execution> |

|  |  |
| --- | --- |
| 14 | </executions> |

|  |  |
| --- | --- |
| 15 | </plugin> |

|  |  |
| --- | --- |
| 16 | </plugins> |

|  |  |
| --- | --- |
| 17 | </build> |

这样Java代码编译时这个简单Mojo就会执行。更多绑定Mojo到生命周期阶段的信息，参考 [Build Lifecycle](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html)。

### Mojo原型

利用Mojo原型创建新插件项目：

|  |  |
| --- | --- |
| 1 | mvn archetype:generate \ |

|  |  |
| --- | --- |
| 2 | -DgroupId=sample.plugin \ |

|  |  |
| --- | --- |
| 3 | -DartifactId=hello-maven-plugin \ |

|  |  |
| --- | --- |
| 4 | -DarchetypeGroupId=org.apache.maven.archetypes \ |

|  |  |
| --- | --- |
| 5 | -DarchetypeArtifactId=maven-archetype-plugin |

### 参数

不带参数的Mojo一般没太大用处，参数提供了以下重要功能：

* 提供了让用户调整插件操作以适应应用的钩子
* 提供一种不需要导航到对象就能提取POM元素值的简单方式

#### 定义Mojo参数

定义一个参数很简单，只需要在Mojo中创建一个实例变量并增加恰当的注解。以下是带参数的简单Mojo示例：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* The greeting to display. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter( property = "sayhi.greeting", defaultValue = "Hello World!" ) |

|  |  |
| --- | --- |
| 5 | **private** String greeting; |

注解之前的部分是参数说明。参数注解将变量声明为Mojo参数。注解的defaultValue参数定义了变量的默认值。这个值可以包含引用项目变量的表达式，如”{project.version}”（在  [“Parameter Expressions” document](http://maven.apache.org/ref/current/maven-core/apidocs/org/apache/maven/plugin/PluginParameterExpressionEvaluator.html)能看到更多）。property参数支持引用用户在命令行中通过-D指定的系统属性。

#### 在项目中配置参数

作为插件定义的一部分，插件的参数值配置在Maven项目的pom.xml中完成，示例：

|  |  |
| --- | --- |
| 1 | <plugin> |

|  |  |
| --- | --- |
| 2 | <groupId>sample.plugin</groupId> |

|  |  |
| --- | --- |
| 3 | <artifactId>hello-maven-plugin</artifactId> |

|  |  |
| --- | --- |
| 4 | <version>1.0-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 5 | <configuration> |

|  |  |
| --- | --- |
| 6 | <greeting>Welcome</greeting> |

|  |  |
| --- | --- |
| 7 | </configuration> |

|  |  |
| --- | --- |
| 8 | </plugin> |

在configuration部分，名为”greeting”的元素名是参数的名称，元素的值”Welcome”是赋予参数的值。  
注意：更多细节参见 [Guide to Configuring Plugins](https://maven.apache.org/guides/mini/guide-configuring-plugins.html)。

#### 单值参数类型

下面是可作为Mojo参数的简单变量类型及其值在pom中的解释规则。

##### Boolean

包括boolean和Boolean。读配置时，读到true字符串时将参数设为true，读到其它字符串将参数设为false。例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My boolean. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** **boolean** myBoolean; |

|  |  |
| --- | --- |
| 1 | <myBoolean>true</myBoolean> |

##### 定点数

包括byte, Byte, int, Integer, long, Long, short和Short。读配置时，xml中的文本通过Integer.parseInt()或对应类型的valueOf()方法转为整型数。字符串必须是有效的十进制整数，只包含数字0到9和可选的前置“-”表示负数。例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Integer. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** Integer myInteger; |

|  |  |
| --- | --- |
| 1 | <myInteger>10</myInteger> |

##### 浮点数

包含double, Double, float和Float。读配置时，使用对应类型的valueOf()将xml中的文本转为二进制。文本可以是Java语言规范3.10.2节规定的任意格式。有效值示例如1.0和6.02E+23。

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Double. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** Double myDouble; |

|  |  |
| --- | --- |
| 1 | <myDouble>1.0</myDouble> |

##### 日期

包括Date类型。读配置时，xml中的文本按”yyyy-MM-dd HH:mm:ss.S a”（如”2005-10-06 2:22:55.1 PM”）或”yyyy-MM-dd HH:mm:ssa”（如”2005-10-06 2:22:55PM”）格式转换。注意解析通过DateFormat.parse()完成，允许格式的少许偏差。这个方法尽力解析出日期和时间，即使格式不严格地匹配上述格式，例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Date. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** Date myDate; |

|  |  |
| --- | --- |
| 1 | <myDate>2005-10-06 2:22:55.1 PM</myDate> |

##### 文件和目录

包括File类。读配置时，xml文件中的文本被当做需要的文件或目录路径。如果是相对路径（不以/或驱动器字母如C:开头），路径是相对于POM所在目录。例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My File. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** File myFile; |

|  |  |
| --- | --- |
| 1 | <myFile>c:\temp</myFile> |

##### URL

包含URL类。读配置时，xml中的文本被当做URL。格式必须遵循RFC2396，看起来像web浏览器的URL(scheme://host:port/path/to/file)。转换URL时对它的任何部分没有限制。

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My URL. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** URL myURL; |

|  |  |
| --- | --- |
| 1 | <myURL><http://maven.apache.org</myURL>> |

##### 普通文本

包含char, Character, StringBuffer和String。读配置时，xml文件中的文本被当做赋予参数的值。对char和Character参数，只使用文本的第一个字符。

##### 枚举

也可以使用枚举类型变量。首先需要定义枚举类型，然后就可以在参数定义中使用了。

|  |  |
| --- | --- |
| 01 | **public** **enum** Color { |

|  |  |
| --- | --- |
| 02 | GREEN, |

|  |  |
| --- | --- |
| 03 | RED, |

|  |  |
| --- | --- |
| 04 | BLUE |

|  |  |
| --- | --- |
| 05 | } |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | /\*\* |

|  |  |
| --- | --- |
| 08 | \* My Enum |

|  |  |
| --- | --- |
| 09 | \*/ |

|  |  |
| --- | --- |
| 10 | @Parameter |

|  |  |
| --- | --- |
| 11 | **private** Color myColor; |

在pom配置中可以使用这样的枚举：

|  |  |
| --- | --- |
| 1 | <myColor>GREEN</myColor> |

也可以使用枚举类型的元素作为默认值，像下面这样：

|  |  |
| --- | --- |
| 01 | **public** **enum** Color { |

|  |  |
| --- | --- |
| 02 | GREEN, |

|  |  |
| --- | --- |
| 03 | RED, |

|  |  |
| --- | --- |
| 04 | BLUE |

|  |  |
| --- | --- |
| 05 | } |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | /\*\* |

|  |  |
| --- | --- |
| 08 | \* My Enum |

|  |  |
| --- | --- |
| 09 | \*/ |

|  |  |
| --- | --- |
| 10 | @Parameter(defaultValue = "GREEN") |

|  |  |
| --- | --- |
| 11 | **private** Color myColor; |

#### 多值参数类型

下面介绍可以在Mojo中作为参数使用的各种复合对象，及它们在POM中的解释规则。一般来说，保存参数值（和参数值中的元素）的对象的类型是按以下步骤决定的（确定有效类型的第一步）：  
1. 如果XML元素包含implementation属性，就使用它  
2. 如果XML标签包含”.”，尝试解析为类的全名  
3. 尝试将XML标签（首字母大写）解析为Mojo所在包中的类  
4. 对数组，使用数组的元素类型（例如，对String[]参数使用String类型）；对集合和映射，使用Mojo配置中指定的类；使用String作为集合类型的entry和map的值的类型。

一旦元素的类型确定了，xml中的文本将被转换为恰当的对象。

##### 数组

数组参数通过多次指定来配置，例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Array. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** String[] myArray; |

|  |  |
| --- | --- |
| 1 | <myArray> |

|  |  |
| --- | --- |
| 2 | <param>value1</param> |

|  |  |
| --- | --- |
| 3 | <param>value2</param> |

|  |  |
| --- | --- |
| 4 | </myArray> |

##### 集合

这一类包含所有实现java.util.Collection的类，如ArrayList或HashSet。这些参数也像数组一样通过多次指定来配置，例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My List. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** List myList; |

|  |  |
| --- | --- |
| 1 | <myList> |

|  |  |
| --- | --- |
| 2 | <param>value1</param> |

|  |  |
| --- | --- |
| 3 | <param>value2</param> |

|  |  |
| --- | --- |
| 4 | </myList> |

关于独立集合元素的映射细节，查看 [Mapping Lists](https://maven.apache.org/guides/mini/guide-configuring-plugins.html" \l "Mapping_Lists).。

##### 映射

这一类包含所有实现java.util.Map而未实现java.util.Properties的类，如HashMap。这些参数通过包含XML标签来配置，形如<key>value</key>，例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Map. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** Map myMap; |

|  |  |
| --- | --- |
| 1 | <myMap> |

|  |  |
| --- | --- |
| 2 | <key1>value1</key1> |

|  |  |
| --- | --- |
| 3 | <key2>value2</key2> |

|  |  |
| --- | --- |
| 4 | </myMap> |

Properties  
这一类包含所有实现java.util.Properties的映射。这些参数通过包含XML标签配置，形如<property><name>myName</name> <value>myValue</value> </property>，例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Properties. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** Properties myProperties; |

|  |  |
| --- | --- |
| 01 | <myProperties> |

|  |  |
| --- | --- |
| 02 | <property> |

|  |  |
| --- | --- |
| 03 | <name>propertyName1</name> |

|  |  |
| --- | --- |
| 04 | <value>propertyValue1</value> |

|  |  |
| --- | --- |
| 05 | <property> |

|  |  |
| --- | --- |
| 06 | <property> |

|  |  |
| --- | --- |
| 07 | <name>propertyName2</name> |

|  |  |
| --- | --- |
| 08 | <value>propertyValue2</value> |

|  |  |
| --- | --- |
| 09 | <property> |

|  |  |
| --- | --- |
| 10 | </myProperties> |

其它对象类型  
这一类包含所有未实现java.util.Map、java.util.Collection和java.util.Dictionary的类。例如：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* My Object. |

|  |  |
| --- | --- |
| 3 | \*/ |

|  |  |
| --- | --- |
| 4 | @Parameter |

|  |  |
| --- | --- |
| 5 | **private** MyObject myObject; |

|  |  |
| --- | --- |
| 1 | <myObject> |

|  |  |
| --- | --- |
| 2 | <myField>test</myField> |

|  |  |
| --- | --- |
| 3 | </myObject> |

查看 [Mapping Complex Objects](https://maven.apache.org/guides/mini/guide-configuring-plugins.html" \l "Mapping_Complex_Objects)了解更多配置这种参数的策略细节。

### 使用setters

如果你想在Maven上下文之外重用Mojo，私有属性映射是好选择，对它的使用没有限制。按上面的示例我们可以使用下划线风格命名私有属性，并提供setter供配置映射机制使用，Mojo看起来是这样的：

|  |  |
| --- | --- |
| 01 | **public** **class** MyQueryMojo |

|  |  |
| --- | --- |
| 02 | **extends** AbstractMojo |

|  |  |
| --- | --- |
| 03 | { |

|  |  |
| --- | --- |
| 04 | @Parameter(property="url") |

|  |  |
| --- | --- |
| 05 | **private** String \_url; |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | @Parameter(property="timeout") |

|  |  |
| --- | --- |
| 08 | **private** **int** \_timeout; |

|  |  |
| --- | --- |
| 09 |  |

|  |  |
| --- | --- |
| 10 | @Parameter(property="options") |

|  |  |
| --- | --- |
| 11 | **private** String[] \_options; |

|  |  |
| --- | --- |
| 12 |  |

|  |  |
| --- | --- |
| 13 | **public** **void** setUrl( String url ) |

|  |  |
| --- | --- |
| 14 | { |

|  |  |
| --- | --- |
| 15 | \_url = url; |

|  |  |
| --- | --- |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |

|  |  |
| --- | --- |
| 18 | **public** **void** setTimeout( **int** timeout ) |

|  |  |
| --- | --- |
| 19 | { |

|  |  |
| --- | --- |
| 20 | \_timeout = timeout; |

|  |  |
| --- | --- |
| 21 | } |

|  |  |
| --- | --- |
| 22 |  |

|  |  |
| --- | --- |
| 23 | **public** **void** setOptions( String[] options ) |

|  |  |
| --- | --- |
| 24 | { |

|  |  |
| --- | --- |
| 25 | \_options = options; |

|  |  |
| --- | --- |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |

|  |  |
| --- | --- |
| 28 | **public** **void** execute() |

|  |  |
| --- | --- |
| 29 | **throws** MojoExecutionException |

|  |  |
| --- | --- |
| 30 | { |

|  |  |
| --- | --- |
| 31 | ... |

|  |  |
| --- | --- |
| 32 | } |

|  |  |
| --- | --- |
| 33 | } |

注意当属性名和插件配置中实际的参数名不一致时，对参数对应的属性名的标记，这是用来告知Maven要使用的getter和setter。