# 编写Maven插件

## 1.1 编写Maven插件的一般步骤：

1. 创建一个maven-plugin项目：插件本身也是Maven项目，特殊的地方在于它的packaging必须是maven-plugin，用户可以使用maven-archetype-plugin快速创建一个Maven插件项目。
2. 为插件编写目标：每个插件都必须包含一个或多个目标，Maven称之为Mojo（与POJO对应，后者指Plain Old Java Object，这里指Maven Old Java Object）。编写插件的时候必须提供一个或者多个继承自AbstractMojo的类。
3. 为目标提供配置点：大部分Maven插件及其目标都是可配置的，因此在编写Mojo的时候需要注意提供可配置的参数。
4. 编写代码实现目标行为：根据实际的需要实现Mojo。
5. 错误处理日志：当Mojo发生异常时，根据情况控制Maven的运行状态。在代码中编写必要的日志以便为用户提供足够的信息。
6. 测试插件：编写自动化的测试代码测试行为，然后再实际运行插件以验证其行为。

## 1.2 案例：编写一个用于代码行统计的Maven插件

本小节详细演示如何实际编写一个简单的用于代码行统计的Maven插件。使用该插件，用户可以了解到Maven项目中各个源代码目录下文件的数量，以及它们加起来共有多少代码行。

要创建一个Maven插件项目，首先使用maven-archetype-plugin骨架(模板)生成模板代码：

|  |
| --- |
| mvn archetype:generate -DarchetypeArtifactId=maven-archetype-plugin -DgroupId=com.chenming -DartifactId=Hello-plugin -Dversion=1.1-SNAPSHOT -DinteractiveMode=false -DarchetypeCatalog=internal |

执行上面的命令，就可以使用maven-archetype-plugin模板创建一个Maven插件工程。

打开项目的pom.xml文件，可以看到如下代码清单所示的内容：



Maven插件项目的POM有两个特殊的地方：

1. 它的packaging必须为maven-plugin，这种特殊的打包类型能控制Maven为其在生命周期阶段绑定插件处理相关的目标，例如在compile阶段，Maven需要为插件项目构建一个特殊插件描述符文件。
2. 从上述代码中可以看到一个artifactId为maven-plugin-api的依赖，该依赖中包含了插件开发所必须的类，例如稍后会看到的AbstractMojo。需要注意的是，上面的代码清单中并没有使用默认Archetype生成的maven-plugin-api版本，而是升级到了3.0，这样做的目的是与Maven的版本保持一致。

插件项目创建好之后，下一步是为插件编写目标。使用Archetype生成的插件项目包含了一个名为MyMojo的java文件，我们将其删除，然后自己创建一个CountMojo，如下面的代码清单所示：

|  |
| --- |
| public class CountMojo extends AbstractMojo {   private static final String[] *INCLUDES\_DEFAULT* = {"java","xml","properties"};   */\*\*  \* @parameter expression="${project.basedir}"  \* @required  \* @readonly  \*/* private File basedir;   */\*\*  \* @parameter expression="${project.build.sourceDirectory}"  \* @required  \* @readonly  \*/* private File sourceDirectory;   */\*\*  \* @parameter expression="${project.build.testSourceDirectory}"  \* @required  \* @readonly  \*/* private File testSourceDirectory;   */\*\*  \* @parameter expression="${project.build.resources}"  \* @required  \* @readonly  \*/* private List<Resource> resources;   */\*\*  \* @parameter expression="${project.build.testResources}"  \* @required  \* @readonly  \*/* private List<Resource> testResources;   */\*\*  \* the file type which will be included for counting  \*  \* @parameter  \*/* private String[] includes;   @Override  public void execute() throws MojoExecutionException, MojoFailureException {  if (includes == null || includes.length == 0) {  includes = *INCLUDES\_DEFAULT*;  }  try {  countDir(sourceDirectory);  countDir(testSourceDirectory);  for (Resource resource : resources) {  countDir(new File(resource.getDirectory()));  }  for (Resource resource : testResources) {  countDir(new File(resource.getDirectory()));  }  } catch (IOException e) {  throw new RuntimeException("Unable to count lines of code",e);  }  }   private void countDir(File dir) throws IOException {  if (!dir.exists()) {  return;  }  List<File> collected = new ArrayList<>();  collectedFiles(collected,dir);  int lines = 0;  for (File sourceFile : collected) {  lines += countLine(sourceFile);  }  }   private void collectedFiles(List<File> collected,File file) {  if (file.isFile()) {  for (String include : includes) {  if (file.getName().endsWith("." + include)) {  collected.add(file);  break;  }  }  } else {  for (File sub : file.listFiles()) {  collectedFiles(collected,sub);  }  }  }   private int countLine(File file) throws IOException {  BufferedReader bufferedReader = new BufferedReader(new FileReader(file));  int lines = 0;  try {  while (bufferedReader.ready()) {  bufferedReader.readLine();  lines++;  }  } finally {  bufferedReader.close();  }  return lines;  } } |

这里简单解释一下：

1. CountMojo统计Java和SQL文件，而不是默认的Java、XML和Properties。
2. 代码中还包含了basedir、sourceDirectory、testSourceDirectory等字段，它们都是用了@parameter标注，但同时关键字expression表示从系统属性读取这几个字段的值。${project.build.testSourceDirectory}，${project.build.sourceDirectory}，${project.basedir}等表达式读者应该已经很熟悉了，它们分别代表了项目的基础目录、主代码目录和测试代码目录。@readonly标注表示不允许用户对其进行配置，因为对于一个项目来说，这几个目录位置都是固定的。
3. execute方法中大家可以看到这样一些信息：如果用户没有配置includes则就是默认的统计包含配置，然后再分别统计、测试代码目录、主资源目录、以及测试资源目录。这里涉及一个countDir()方法。