InfoQ Maven专栏（七）——常用Maven插件介绍（上）

April 27th, 2011

我们都知道Maven本质上是一个插件框架，它的核心并不执行任何具体的构建任务，所有这些任务都交给插件来完成，例如编译源代码是由maven-compiler-plugin完成的。进一步说，每个任务对应了一个插件目标（goal），每个插件会有一个或者多个目标，例如maven-compiler-plugin的compile目标用来编译位于src/main/java/目录下的主源码，testCompile目标用来编译位于src/test/java/目录下的测试源码。

用户可以通过两种方式调用Maven插件目标。第一种方式是将插件目标与生命周期阶段（lifecycle phase）绑定，这样用户在命令行只是输入生命周期阶段而已，例如Maven默认将maven-compiler-plugin的compile目标与compile生命周期阶段绑定，因此命令**mvn compile**实际上是先定位到compile这一生命周期阶段，然后再根据绑定关系调用maven-compiler-plugin的compile目标。第二种方式是直接在命令行指定要执行的插件目标，例如**mvn archetype:generate** 就表示调用maven-archetype-plugin的generate目标，这种带冒号的调用方式与生命周期无关。

认识上述Maven插件的基本概念能帮助你理解Maven的工作机制，不过要想更高效率地使用Maven，了解一些常用的插件还是很有必要的，这可以帮助你避免一不小心重新发明轮子。多年来Maven社区积累了大量的经验，并随之形成了一个成熟的插件生态圈。Maven官方有两个插件列表，第一个列表的GroupId为org.apache.maven.plugins，这里的插件最为成熟，具体地址为：http://maven.apache.org/plugins/index.html。第二个列表的GroupId为org.codehaus.mojo，这里的插件没有那么核心，但也有不少十分有用，其地址为：http://mojo.codehaus.org/plugins.html。

接下来笔者根据自己的经验介绍一些最常用的Maven插件，在不同的环境下它们各自都有其出色的表现，熟练地使用它们能让你的日常构建工作事半功倍。

maven-antrun-plugin

http://maven.apache.org/plugins/maven-antrun-plugin/

maven-antrun-plugin能让用户在Maven项目中运行Ant任务。用户可以直接在该插件的配置以Ant的方式编写Target，然后交给该插件的run目标去执行。在一些由Ant往Maven迁移的项目中，该插件尤其有用。此外当你发现需要编写一些自定义程度很高的任务，同时又觉得Maven不够灵活时，也可以以Ant的方式实现之。maven-antrun-plugin的run目标通常与生命周期绑定运行。

maven-archetype-plugin

http://maven.apache.org/archetype/maven-archetype-plugin/

Archetype指项目的骨架，Maven初学者最开始执行的Maven命令可能就是**mvn archetype:generate**，这实际上就是让maven-archetype-plugin生成一个很简单的项目骨架，帮助开发者快速上手。可能也有人看到一些文档写了**mvn archetype:create**，但实际上create目标已经被弃用了，取而代之的是generate目标，该目标使用交互式的方式提示用户输入必要的信息以创建项目，体验更好。maven-archetype-plugin还有一些其他目标帮助用户自己定义项目原型，例如你有一个产品需要交付给很多客户进行二次开发，你就可以为他们提供一个Archtype，帮助他们快速上手。

maven-assembly-plugin

http://maven.apache.org/plugins/maven-assembly-plugin/

maven-assembly-plugin的用途是制作项目分发包，该分发包可能包含了项目的可执行文件、源代码、readme、平台脚本等等。maven-assembly-plugin支持各种主流的格式如zip、tar.gz、jar和war等，具体打包哪些文件是高度可控的，例如用户可以按文件级别的粒度、文件集级别的粒度、模块级别的粒度、以及依赖级别的粒度控制打包，此外，包含和排除配置也是支持的。maven-assembly-plugin要求用户使用一个名为assembly.xml的元数据文件来表述打包，它的single目标可以直接在命令行调用，也可以被绑定至生命周期。

maven-dependency-plugin

http://maven.apache.org/plugins/maven-dependency-plugin/

maven-dependency-plugin最大的用途是帮助分析项目依赖，**dependency:list**能够列出项目最终解析到的依赖列表，**dependency:tree**能进一步的描绘项目依赖树，**dependency:analyze**可以告诉你项目依赖潜在的问题，如果你有直接使用到的却未声明的依赖，该目标就会发出警告。maven-dependency-plugin还有很多目标帮助你操作依赖文件，例如**dependency:copy-dependencies**能将项目依赖从本地Maven仓库复制到某个特定的文件夹下面。

maven-enforcer-plugin

http://maven.apache.org/plugins/maven-enforcer-plugin/

在一个稍大一点的组织或团队中，你无法保证所有成员都熟悉Maven，那他们做一些比较愚蠢的事情就会变得很正常，例如给项目引入了外部的SNAPSHOT依赖而导致构建不稳定，使用了一个与大家不一致的Maven版本而经常抱怨构建出现诡异问题。maven-enforcer-plugin能够帮助你避免之类问题，它允许你创建一系列规则强制大家遵守，包括设定Java版本、设定Maven版本、禁止某些依赖、禁止SNAPSHOT依赖。只要在一个父POM配置规则，然后让大家继承，当规则遭到破坏的时候，Maven就会报错。除了标准的规则之外，你还可以扩展该插件，编写自己的规则。maven-enforcer-plugin的enforce目标负责检查规则，它默认绑定到生命周期的validate阶段。

maven-help-plugin

http://maven.apache.org/plugins/maven-help-plugin/

maven-help-plugin是一个小巧的辅助工具，最简单的**help:system**可以打印所有可用的环境变量和Java系统属性。**help:effective-pom**和**help:effective-settings**最为有用，它们分别打印项目的有效POM和有效settings，有效POM是指合并了所有父POM（包括Super POM）后的XML，当你不确定POM的某些信息从何而来时，就可以查看有效POM。有效settings同理，特别是当你发现自己配置的settings.xml没有生效时，就可以用**help:effective-settings**来验证。此外，maven-help-plugin的describe目标可以帮助你描述任何一个Maven插件的信息，还有all-profiles目标和active-profiles目标帮助查看项目的Profile。

maven-release-plugin

http://maven.apache.org/plugins/maven-release-plugin/

maven-release-plugin的用途是帮助自动化项目版本发布，它依赖于POM中的SCM信息。**release:prepare**用来准备版本发布，具体的工作包括检查是否有未提交代码、检查是否有SNAPSHOT依赖、升级项目的SNAPSHOT版本至RELEASE版本、为项目打标签等等。**release:perform**则是签出标签中的RELEASE源码，构建并发布。版本发布是非常琐碎的工作，它涉及了各种检查，而且由于该工作仅仅是偶尔需要，因此手动操作很容易遗漏一些细节，maven-release-plugin让该工作变得非常快速简便，不易出错。maven-release-plugin的各种目标通常直接在命令行调用，因为版本发布显然不是日常构建生命周期的一部分。

本文已经首发于InfoQ中文站，版权所有，原文为《Maven实战（七）——常用Maven插件介绍（上）》

原创文章，转载请注明出处，本文地址：http://www.juvenxu.com/2011/04/27/infoq-maven-most-used-maven-plugins-a/