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/

Archtype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，例如你由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。

maven-resources-plugin

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

为了使项目结构更为清晰，Maven区别对待Java代码文件和资源文件，maven-compiler-plugin用来编译Java代码，maven-resources-plugin则用来处理资源文件。默认的主资源文件目录是`src/main/resources`，很多用户会需要添加额外的资源文件目录，这个时候就可以通过配置maven-resources-plugin来实现。此外，资源文件过滤也是Maven的一大特性，你可以在资源文件中使用${propertyName}形式的Maven属性，然后配置maven-resources-plugin开启对资源文件的过滤，之后就可以针对不同环境通过命令行或者Profile传入属性的值，以实现更为灵活的构建。

maven-surefire-plugin

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

可能是由于历史的原因，Maven 2/3中用于执行测试的插件不是maven-test-plugin，而是maven-surefire-plugin。其实大部分时间内，只要你的测试类遵循通用的命令约定（以Test结尾、以TestCase结尾、或者以Test开头），就几乎不用知晓该插件的存在。然而在当你想要跳过测试、排除某些测试类、或者使用一些TestNG特性的时候，了解maven-surefire-plugin的一些配置选项就很有用了。例如 mvn test -Dtest=FooTest 这样一条命令的效果是仅运行FooTest测试类，这是通过控制maven-surefire-plugin的test参数实现的。

build-helper-maven-plugin

http://mojo.codehaus.org/build-helper-maven-plugin/

Maven默认只允许指定一个主Java代码目录和一个测试Java代码目录，虽然这其实是个应当尽量遵守的约定，但偶尔你还是会希望能够指定多个源码目录（例如为了应对遗留项目），build-helper-maven-plugin的add-source目标就是服务于这个目的，通常它被绑定到默认生命周期的generate-sources阶段以添加额外的源码目录。需要强调的是，这种做法还是不推荐的，因为它破坏了 Maven的约定，而且可能会遇到其他严格遵守约定的插件工具无法正确识别额外的源码目录。

build-helper-maven-plugin的另一个非常有用的目标是attach-artifact，使用该目标你可以以classifier的形式选取部分项目文件生成附属构件，并同时install到本地仓库，也可以deploy到远程仓库。

exec-maven-plugin

http://mojo.codehaus.org/exec-maven-plugin/

exec-maven-plugin很好理解，顾名思义，它能让你运行任何本地的系统程序，在某些特定情况下，运行一个Maven外部的程序可能就是最简单的问题解决方案，这就是exec:exec的用途，当然，该插件还允许你配置相关的程序运行参数。除了exec目标之外，exec-maven-plugin还提供了一个java目标，该目标要求你提供一个mainClass参数，然后它能够利用当前项目的依赖作为classpath，在同一个JVM中运行该mainClass。有时候，为了简单的演示一个命令行Java程序，你可以在POM中配置好exec-maven-plugin的相关运行参数，然后直接在命令运行 mvn exec:java 以查看运行效果。

jetty-maven-plugin

http://wiki.eclipse.org/Jetty/Feature/Jetty_Maven_Plugin

在进行Web开发的时候，打开浏览器对应用进行手动的测试几乎是无法避免的，

这种测试方法通常就是将项目打包成war文件，然后部署到Web容器中，再启动容器进行验证，这显然十分耗时。为了帮助开发者节省时间，jetty-maven-plugin应运而生，它完全兼容 Maven项目的目录结构，能够周期性地检查源文件，一旦发现变更后自动更新到内置的Jetty Web容器中。做一些基本配置后（例如Web应用的contextPath和自动扫描变更的时间间隔），你只要执行 mvn jetty:run ，然后在IDE中修改代码，代码经IDE自动编译后产生变更，再由jetty-maven-plugin侦测到后更新至Jetty容器，这时你就可以直接测试Web页面了。需要注意的是，jetty-maven-plugin并不是宿主于Apache或Codehaus的官方插件，因此使用的时候需要额外的配置`settings.xml`的pluginGroups元素，将org.mortbay.jetty这个pluginGroup加入。

versions-maven-plugin

http://mojo.codehaus.org/versions-maven-plugin/

很多Maven用户遇到过这样一个问题，当项目包含大量模块的时候，为他们集体更新版本就变成一件烦人的事情，到底有没有自动化工具能帮助完成这件事情呢？（当然你可以使用sed之类的文本操作工具，不过不在本文讨论范围）答案是肯定的，versions-maven- plugin提供了很多目标帮助你管理Maven项目的各种版本信息。例如最常用的，命令 mvn versions:set -DnewVersion=1.1-SNAPSHOT 就能帮助你把所有模块的版本更新到1.1-SNAPSHOT。该插件还提供了其他一些很有用的目标，display-dependency- updates能告诉你项目依赖有哪些可用的更新；类似的display-plugin-updates能告诉你可用的插件更新；然后use- latest-versions能自动帮你将所有依赖升级到最新版本。最后，如果你对所做的更改满意，则可以使用 mvn versions:commit 提交，不满意的话也可以使用 mvn versions:revert 进行撤销。

小结

本文介绍了一些最常用的Maven插件，这里指的“常用”是指经常需要进行配置的插件，事实上我们用Maven的时候很多其它插件也是必须的，例如默认的编译插件maven-compiler-plugin和默认的打包插件maven-jar-plugin，但因为很少需要对它们进行配置，因此不在本文讨论范围。了解常用的Maven插件能帮助你事倍功半地完成项目构建任务，反之你就可能会因为经常遇到一些难以解决的问题而感到沮丧。本文介绍的插件基本能覆盖大部分Maven用户的日常使用需要，如果你真有非常特殊的需求，自行编写一个Maven插件也不是难事，更何况还有这么多开放源代码的插件供你参考。

本文的这个插件列表并不是一个完整列表，读者有兴趣的话也可以去仔细浏览一

下Apache和Codehaus Mojo的Maven插件列表，以的到一个更为全面的认识。最后，在线的Maven仓库搜索引擎如http://search.maven.org/也能帮助你快速找到自己感兴趣的Maven插件。