生命周期 -> phase -> plugin goal853769620

mvn clean package，背后的一套原理

现在，引出来一个问题，就是默认情况下，你什么都不搞的话，其实就少数一些phase绑定了一些maven内置的插件的goal，相当于有大量的phase其实是空置的

另外一个问题，可能在我们使用maven的过程中，有很多各种各样其他的一些需求，我给大家举个两个最常见的例子：

（1）比如说，我们打出来的一个jar包，放到生产环境去跑的时候，实际上是需要将所有依赖的jar包都打在这个jar包里面的，这样的话，在实际程序运行的时候，才能找到所有需要的依赖。

默认maven的功能是不支持的，所以此时就需要用到maven其他的插件，就有很多人或者公司，为了增强maven的功能，基于maven实现各种各样酷炫或者实用的功能，于是基于maven的这一套生命周期+插件的机制，开发了各种各样的插件，然后呢，我们如果要使用别人的插件在我们的项目里实现某些特殊的功能，此时就可以在pom.xml里面配置那个插件，包括最重要的，配置那个插件绑定到哪个phase上去。

然后呢，在maven的命令执行的时候，不是默认会执行生命周期中的各种phase吗，如果你绑定了某个第三方的插件到phase，此时插件就会执行，然后实现我们想要的功能。

比如说，我们就可以用assembly的插件，配置在里面，打包的时候，将所有依赖的jar包都放入我们自己工程的jar包里。

（2）另外一个，就是我们可能需要的是，除了能运行单元测试代码以外，我们还需要在单元测试执行完之后，运行一个其他的插件，计算出单元测试对我们所有代码的一个覆盖率。就是说单元测试，测试了百分之多少的代码。此时也需要使用一个第三方的cobertura插件。

（3）比如我们后面会讲解，开发了一个web工程，需要在本地，在eclipse里面随时很轻易的就启动一个web容器，然后部署这个web工程，而不是用比较重型的eclipse+tomcat的那种方式。运行mvn命令，直接web容器也启动了，web工程也部署好了，接着打开浏览器，访问接口，就可以冒烟测试。这种情况下，也需要使用一个专门的jetty插件。绑定到一个phase，phase结束之后，就直接插件启动一个jetty容器，然后部署工程打好的war包到jetty容器中，对外提供访问。

就是百度搜索一下，找到你需要的那个功能对应的插件，然后搜一下那个插件配置的范例，接着在pom.xml里面配置那个插件，将插件绑定到某个phase。此时每次你执行mvn命令的时候，插件就可以被执行，然后提供更多的功能。

来学习如何配置插件，插件配置的语法是什么

学完了以后，我们以后应该彻底了解两个事情：

（1）你配置了插件以后，结合生命周期执行原理，了解插件执行的原理是什么

（2）你完全可以看懂眼花缭乱，乱七八糟的那些所谓的插件的配置

（3）我们这一讲不会去练习配置插件，因为后面插件有的你配置，我们上面讲解的那些功能，后面都是需要借助插件去实现的

1、插件和goal

plugin

每个插件都有多个goal，每个goal都是一个具体的功能

举个例子，dependency插件有十几个goal，可以进行分析项目依赖，列出依赖树，等等

插件的goal，写法就是plugin:goal，比如dependency:tree

用mvn dependency:tree，就可以手动执行某个插件的goal，执行某种功能

<resource>的配置的说明，先简单说一下，后面有一讲，是专门讲解这个资源这一块的，resources:resources，process-resources phase。

2、将插件的goal绑定到phase上

maven内置就会绑定一些插件的goal到phase上，上一讲已经说过了

这里讲解一下，我们如果要使用某个maven插件，如何手动将插件的goal绑定到phase上

比如将source插件的jar-no-fork goal绑定到verify phase，在完成集成测试之后，就生成源码的jar包，这里可以看到绑定plugin的语法：

简单说一句，有不少开源的项目，不仅仅会给你提供下载一个可以立即使用的jar包，还会给你提供一个用src结尾的一个jar包，那个jar包里放的就是那个开源项目的源码。

<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>

运行mvn verify，就可以看到生成了一个包含源码的jar包

即使不配置绑定的phase也可以，因为大多数插件都默认内置了要绑定的phase，比如这个插件就内置绑定在package phase。

3、配置插件

如果在命令行执行插件，可以用-Dkey=value来进行插件的设置

比如mvn install -Dmaven.test.skip=true，就是surefire插件在测试的时候提供的参数，设置为true就会跳过测试

此外也可以在pom.xml中用来配置

<build>

<plugins>

<plugin>

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

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

<version>2.1</version>

<configuration>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

4、找到你需要的插件

在http://maven.apache.org/plugins/index.html中可以找到所有的插件，里面有文档

很简单，很接地气的办法，就是一般你可以百度搜索，用maven启动jetty插件，用maven部署tomcat插件

5、插件解析

先从本地仓库找插件，没有则从远程仓库找插件

插件的远程仓库也需要配置，maven默认配置了远程的插件仓库

<pluginRepositories>

<pluginRepository>

<id>central</id>

<name>Maven Plugin Repository</name>

<url>http://repo1.maven.org/maven2</url>

<layout>default</layout>

<snapshots>

<enabled>false</enabled>

</snapshots>

<releases>

<updatePolicy>never</updatePolicy>

</releases>

</pluginRepository>

</pluginRepositories>