Maven3 实战笔记 04Maven 的生命周期和插件

刘岩

Email:suhuanzheng7784877@163.com

1. Maven 的生命周期

Maven 的生命周期其实是指它对所有的构建过程进行了反复的推敲、反思，之后总结了一套高度抽象过程。这个过程是高度完善的、容易扩展的。基本上包含了项目的清理、初始化、编译、测试、打包、集成测试、验证、部署、、站点生成等步骤，几乎所有的项目生命周期也就这样。Maven 项目周期是一个抽象的概念，这个概念性的东西意味着它并不做任何实质性的事情，也就是说：它就像接口，只定义规范，具体细节它不管。具体的实现细节则交给了 Maven 的各个丰富的插件。Maven 的插件机制有可能是跟 Eclipse 学的，基于一个内核 core，定义一堆流程性的东西，让插件去实现这些规范。其他组织也可以根据这套规范插入自己的东西，形成有特色化的、自定制的 Maven。

Maven 有三套相互独立的生命周期，分别是：clean、default、site。clean 主要是清理项目、default 是 Maven 最核心的的构建项目、site 是生成项目站点。每一个大的生命周期又分为很多个阶段。后面的阶段依赖于前面的阶段，这点有点像 Ant 的构建依赖。生命周期本身相互独立，用户可以仅仅调用生命周期的某一个阶段，也就是说用户调用了 default 周期的任何阶段，并不会触发 clean 周期以及 site 周期的任何事情。

2. Maven 生命周期阶段详解

3 大生命周期蕴含着小小的阶段，我们按顺序看一下

clean 周期：

pre-clean：准备清理

clean：真正的清理工作

post-clean：执行清理后的一些后续工作

default 周期：

validate：验证

initialize：初始化配置

generate-sources：生成源代码编译目录

process-sources：处理项目主资源文件，复制资源文件到 outputclasspath

generate-resources：生成资源目录

process-resources：处理资源文件

complie：编译源代码

process-classes：处理编译后文件

generate-test-sources：生成测试目录

process-test-sources：处理项目测试资源文件，复制测试资源文件到 outputclasspath

generate-test-resources：生成测试资源文件

process-test-resources：处理测试资源文件

test-compile：编译测试代码

process-test-classes:处理测试代码

test：单元测试运行测试代码

prepare-package：打包前的准备

package：将编译好的代码打包成为 jar 或者 war 或者 ear 等等

pre-integration-test：准备整体测试

integration-test：整体测试

post-integration-test：为整体测试收尾

verify：验证

install：安装到本地 Maven 库

deploy：将最终包部署到远程 Maven 仓库

site 周期：

pre-site：准备生成站点

site：生成站点及文档

post-site：站点收尾

site-deploy：将生成的站点发布到服务器上

比如说在命令行执行了

```
mvn clean
```

就是执行到 clean 周期的 clean 阶段。也就是说实际执行了 pre-clean 阶段与 clean 阶段。

```
mvn deploy
```

就是执行了整个 default 生命周期

```
mvn clean deploy site-deploy
```

这个就是执行了 clean 周期的前两个阶段、default 周期的所有阶段、site 周期的所有阶段。

3. Maven 的插件机制

之前我们就说了 Maven 的生命周期仅仅是个抽象的标准，不干实事的，真正干事的人藏在了幕后，就是 Maven 插件。插件本身为了能够代码复用，往往一个插件实现了很多功能，这个如果我们做过 Eclipse 插件开发的人也许更清楚，比如一个 Eclipse 的 SVN 插件，即实现了可以查看远程 SVN 资源库的信息，也可以下载远程代码，还可以上传代码。这实际上是 3 个功能，而由一个 jar 实现。在 Maven 中，管这个叫做“目标”。比如

maven-dependency-plugin 基于项目依赖实现了很多事情，分析依赖、列出依赖树、分析依赖来源等等。每个功能对应着一个插件的目标，插件的目标越多，插件的功能越多。

比如

```
mvn dependency:analyze
```

就是使用 maven-dependency-plugin 插件的 analyze 目标，分析项目的依赖。

```
[WARNING] Unused declared dependencies found:
[WARNING]     org.springframework:spring-core:jar:2.5.6:compile
[WARNING]     org.springframework:spring-beans:jar:2.5.6:compile
```

Maven 的生命周期与 Maven 插件是项目绑定的，Maven 默认地将一些默认插件的目标与 Maven 的生命周期维系在了一起，比如 default 的 compile 这个阶段就是和 maven-compiler-plugin 这个插件的 compile 目标维系着不可分割的关系。前者是领导，复杂发号施令，指定规则，后者是小兵，专门根据任务干活儿的人。为了不让用户不用任何配置就能进行一般程度的项目构建，Maven 默认给自己生命周期的核心阶段绑定了自己的插件。

clean 如下：

生命周期阶段	插件目标
pre-clean	
clean	maven-clean-plugin:clean
post-clean	

site 如下：

生命周期阶段	插件目标
pre-site	

site	maven-site-plugin:site
post-site	
site-deploy	maven-site-plugin:deploy

最麻烦的就是最核心的 default

生命周期阶段	插件目标
process-resources	maven-resources-plugin:resources
compile	maven-compiler-pugin:compile
process-test-resources	maven-resources-plugin:testResources
test-compile	maven-compiler-plugin:testCompile
test	maven-surefire-plugin:testCompile
package	maven-jar-plugin:jar
install	maven-install-plugin:install
deploy	maven-deploy-plugin:deploy

其他没绑定插件的就是说没有什么实际行为。

在我们自己的项目中绑定插件，比如在 pom.xml 内容添加如下内容

```
    <build>
        <resources>
            <resource>
                <directory>src/main/resource</directory>
            </resource>
        </resources>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-source-plugin</artifactId>
                <version>2.1.1</version>
                <executions>
                    <execution>
                        <id>buildSource</id>
```

```
                    <phase>verify</phase>
                    <goals>
                        <goal>jar-no-fork</goal>
                    </goals>
                    <inherited>false</inherited>
                    <configuration>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

之后执行命令

```
mvn verify
```

看到输出文件夹就包含了我们的源代码 source 的 jar。这个打包源代码的“目标”被绑定到了 default 周期的 verify 执行。还有一点就是有些插件一旦写上了 pom.xml 会有默认的绑定周期，比如就拿以上插件说事，如果将<phase>verify</phase>去掉，执行

```
mvn package
```

源代码依然输出，其实它默认适合 default 周期的 package 阶段绑定的。Goals 代表该插件的某些目标（功能）。

插件还能进行全局性质的参数配置，参数是什么就不用多说了吧，大家接触 linux 的都知道吧。Configuration 就是配置参数的。

```
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.1</version>
            <configuration>
                <target>1.5</target>
            </configuration>
        </plugin>
```

4. Maven 插件的详细信息

如果想获取插件的详细信息，一种途径就是通过在线官网查询（google 一下就知道了），

一种就是利用它的另一个插件，maven-help-plugin。比如在命令行输入如下

```
mvn help:describe -D plugin=org.apache.maven.plugins:maven-compiler-plugin:2.1
```

效果如下，显示了一些插件的信息

```
Name: Maven Compiler Plugin

Description: The Compiler Plugin is used to compile the sources of your

   project.

Group Id: org.apache.maven.plugins

Artifact Id: maven-compiler-plugin

Version: 2.1

Goal Prefix: compiler



This plugin has 3 goals:



compiler:compile

   Description: Compiles application sources



compiler:help

   Description: Display help information on maven-compiler-plugin.

      Call

         mvn compiler:help -Ddetail=true -Dgoal=<goal-name>

      to display parameter details.
```

```
compiler:testCompile

   Description: Compiles application test sources.



For more information, run 'mvn help:describe [...] -Ddetail'
```

需要注意的就是 Goal Prefix: compiler 这里，是代表该插件的目标前缀写法，我称之为目标简写，也就是说你可以简写为

```
mvn compiler:compile
```

就可以使用 maven 的 maven-compiler-plugin 插件完成编译项目的功能了。其实使用“插件:目标”的方式是适合该功能不方便与 Maven 生命周期绑定的情况下。

5. 总结

这次主要概括了 Maven 的生命周期以及它的插件机制和插件的使用。生命周期是 Maven 核心的东西，插件也是 Maven 核心的东西，所以还是有必要看看的。下次我们单独来看看之前没提到的解析机制，包括 Maven 仓库的依赖解析和 Maven 插件的解析机制。