# Maven生命周期，插件，聚合，继承

## 一 生命周期

### 1.1 何为生命周期

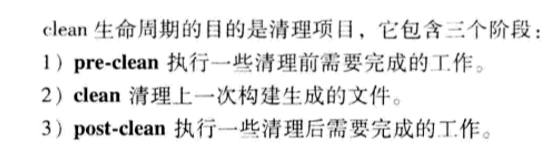
Maven从大量项目中学习与反思，然后总结了一套高度完善的，易于扩展的的生命周期。这个生命周期包含了项目的清理，初始化，编译，测试，打包，集成测试，验证，部署和站点生成等几乎所有构建步奏。每一个步奏可以绑定一个或者多个插件行为。而且maven为大多数构建步骤编写并绑定了默认的插件。

### 1.2 生命周期详解

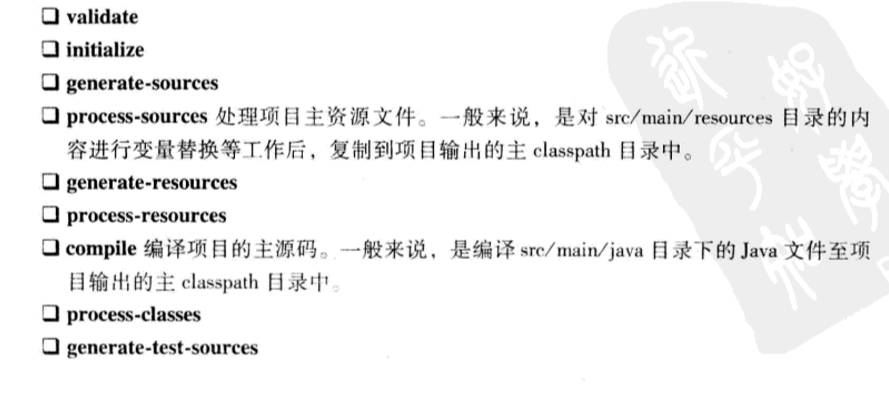
Maven拥有三套互相独立的生命周期,他们分别为clean.default.和site。Clean生命周期的目的是清理项目，Default生命周期的目的构建项目，site项目的生命周期是建立项目站点。

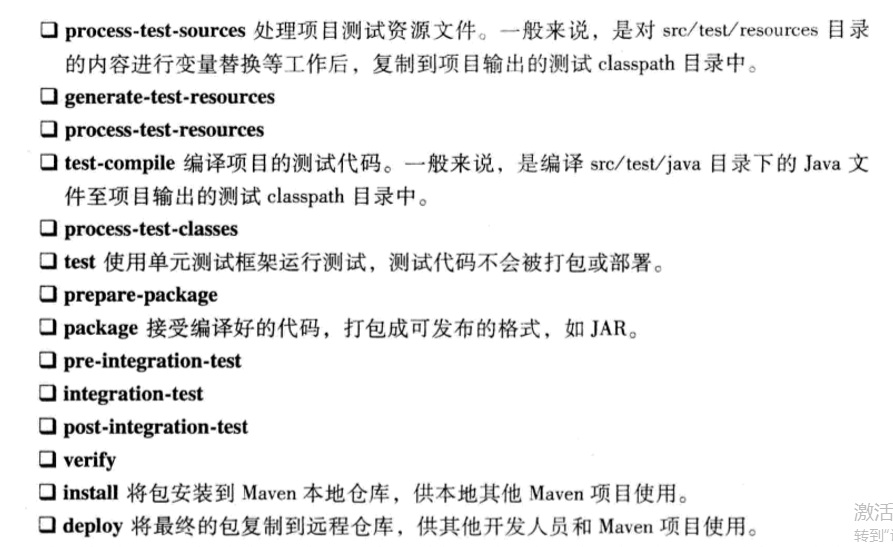
每个生命周期包含一些阶段，这些阶段是有顺序的，并且后面的阶段依赖于前面的阶段，用户和maven最直接的交互就是调用maven生命周期中的这些阶段。

#### 1.2.1 clean生命周期阶段

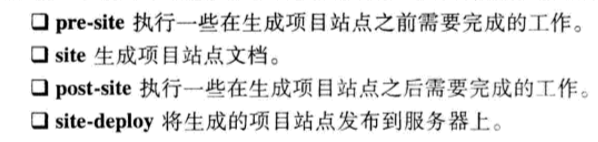


#### 1.2.2 default生命周期阶段





#### 1.2.3 site生命周期阶段



实际上maven的主要生命周期阶段都差不多，常用的命令都是基于这些阶段组合而成，因此我们只需要对maven的这些生命周期有一个基本的了解就可以。

## 二 插件

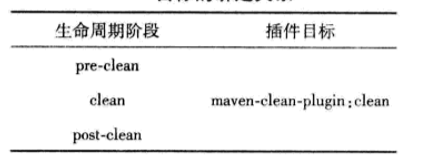
### 2.1 插件绑定

Maven的核心仅仅定义抽象的生命周期，具体的任务是教育插件完成的。插件以独立的构建形式存在。因此maven的核心分发包，只有不到3m大小。Maven会在需要的时候下载并使用插件。Maven的插件与生命周期相互绑定，以完成实际的构建目标。具体而言，是生命周期的阶段与插件的目标相互绑定，已完成具体的构建任务。

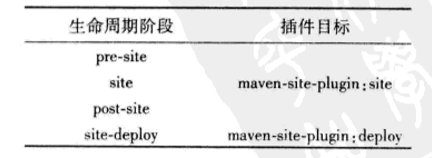
### 2.2 插件内置绑定

为了能让用户不用配置就能构建项目，maven在核心为一些主要的生命周期阶段绑定了很多插件目标。当用户通过命令行调用生命周期阶段时，对应的插件目标就会执行相应的任务。

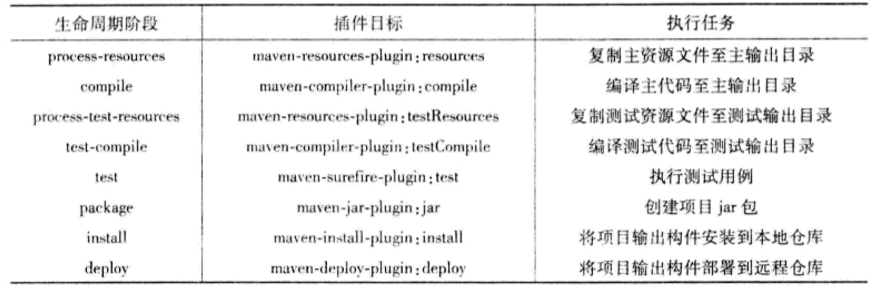
#### 2.2.1 clean



#### 2.2.2 sit



#### 2.2.3 default



### 2.3 自定义绑定

<plugin>

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

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

<version>3.2</version>

<executions>

<execution>

<id>compiler-id</id>

<phase>compile</phase>

<goals>

xxxx

</goals>

</execution>

</executions>

<configuration>

<source>1.8</source>

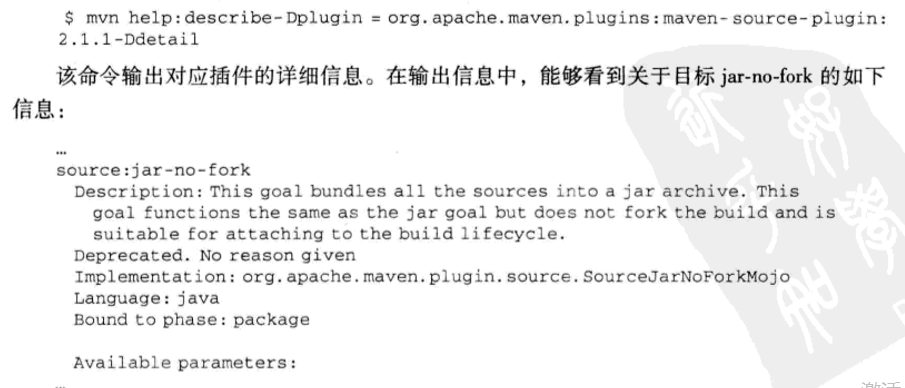
<target>1.8</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

如上片段所示的一段代码，原本是一个插件，其实它已经在内部默认绑定到了compile阶段了，但是出于个别原因我在这里做详解，如果是一个先编写好的插件，那么我们可以让它绑定到指定的阶段，通过配置executions元素。其实在编写插件的时候很多插件是早已经绑定了生命周期的不再需要我们配置了我们可以用使用。



这样我们就可以知道这个插件是否已经绑定了生命周期了。当插件目标被绑定到不同的生命周期阶段的时候，其执行的顺序由生命周期阶段的先后顺序决定。如果多个目标被绑定到同一个阶段，这些插件生明的先后顺序决定了插件的执行顺序。

<plugin>

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

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

<version>3.2</version>

<executions>

<execution>

<id>compiler-id</id>

<phase>validate</phase>

<goals>

run

</goals>

<configuration>

<testIncludes>

</testIncludes>

</configuration>

</execution>

<execution>

<id>compiler-id</id>

<phase>compile</phase>

<goals>

run

</goals>

<configuration>

<compilerArgs></compilerArgs>

</configuration>

</execution>

</executions>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

如上面的配置所示configeration为这个插件的全局配置，可以当做这个插件的configeration中配置了不同的任务，在生命周期的不同阶段执行不同的任务。

### 2.4 插件解析机制

Maven不需要提供完整的插件坐标信息，就可以解析得正确的插件,maven的这一特性是一把双忍剑。虽然它简化了插件的使用和配置，可一旦插件的行为出现异常，用户是很难定位到问题的插件构件的。Maven插件与maven依赖都有远程仓库同步的机制。



## 三 聚合与继承

### 3.1 聚合与聚合配置

出于我们一个要构建多个项目的原因，或者说我们要模块化构建项目的原因，我们采用聚合的方式引入项目

<modules>

<module>../yon-team</module>

<module>yon-jars</module>

<module>../yon-msg</module>

<module>yon-cache</module>

<module>yon-report</module>

</modules>

### 3.2 继承与继承配置

聚合解决了多模块同时构建的问题，那么继承解决了什么问题，当场是复用的问题了，比如依赖的版本号，依赖的jar包，这些每一个模块都共有的配置，就没必要在每一个模块都配置了，我们希望它能够配置到一个单独的父模块中去，其他的模块都继承自这个父模块。

<parent>

<groupId>com.yon</groupId>

<artifactId>yon-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

<relativePath>../yon-parent</relativePath>

</parent>

<artifactId>yon-team</artifactId>

<packaging>war</packaging>

#### 3.2.1 继承之依赖管理

<dependencyManagement>

<dependency>

<groupId>javax.mail</groupId>

<artifactId>mail</artifactId>

<version>${mail.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.jaxrs</groupId>

<artifactId>jackson-jaxrs-json-provider</artifactId>

<version>${cxf.version}</version>

</dependency>

</dependencies>

</dependencyManagement

如上所示在父目录中定义了，依赖的统一描述，包括版本号,到了子项目中我就可以继承以后进行简化，提高复用能力

<dependencies>

<dependency>

<groupId>javax.mail</groupId>

<artifactId>mail</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.jaxrs</groupId>

<artifactId>jackson-jaxrs-json-provider</artifactId>

</dependency>

</dependencies>

#### 3.2.2 继承之插件管理

<build>

<pluginManagement>

<plugins>

<plugin>

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

<artifactId>maven-resources-plugin</artifactId>

<version>2.7</version>

<configuration>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- java编译插件 -->

<plugin>

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

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

<version>3.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

如上片段所示我们可以在父工程中定义好插件的基本配置，而在子工程中只需要简单的声明。

<build>

<finalName>${project.artifactId}</finalName>

<plugins>

<!-- 资源文件拷贝插件 -->

<plugin>

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

<artifactId>maven-resources-plugin</artifactId>

</plugin>

<!-- java编译插件 -->

<plugin>

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

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

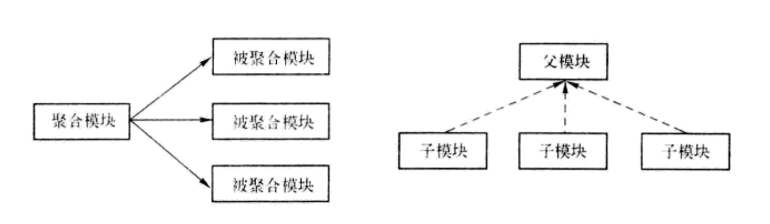
</plugin>

</plugins>

以上父工程中的配置，都不会调用父工程中的行为，只是一种声明，然而。

### 3.3 聚合与继承关系

对于聚合来说他知道有哪些聚合模块，但那些被聚合的模块不知道有这个聚合模块的存在。对于继承来说，它不知道子模块继承了它，但那些子pom都必须知道自己的父pom是谁。硬要说他们的关系，那么就是聚合模块必须是pom包类型，当然被聚合模块的父模块也必须是pom的包类型。



### 3.4反应堆

反应堆是指所有模块组成的一个构建结构。对于单模块的项目反应堆就是他本身。对于多模块的项目反应堆就是模块之间继承与依赖的关系。从而能够自动计算出合理的模块顺序。模块的依赖关系将会反应堆构成一个有向非循环图（DAG）图。各个模块是该图的节点，依赖关系构成了有向边，这个图不可以出现环。出现环结构是maven就会报错。