1. Maven 是什么
   1. 绝大多数Maven用户称它是一个构建工具，用来把源代码构建成可发布的构件的工具。
   2. 项目管理工具，除了构建的功能，maven还可以生成报告，生成站点。
   3. Maven是一个项目管理工具，包含一个项目对象模型(Project Object Model) ， 一组标准集合，一个项目生命周期(Project Lifecycle) ，一个依赖管理系统( Dependency Management System)， 和用来运行定义再生命周期阶段(phase) 的插件(plugin)目标(goal)的逻辑。 当你用一个明确定义的项目对象模型来描述你的项目，然后maven就可以应用横切的逻辑，这些逻辑来自一组共享的或者自定义的插件。
2. 约定优于配置

约定优于配置是个简单概念，系统，框架应该假定合理的默认值，而非要求提供不必要的配置。

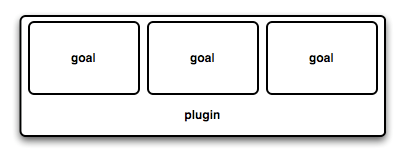
1. maven 为构建软件提供了一个一般的接口。
2. 基于Maven插件的全局性重用，maven核心除了解析xml文档，管理生命周期与插件之外什么也不做。Maven被设计成将主要职责委派给一组maven插件，这些插件可以影响maven生命周期，提供对目标的访问。
3. Maven help 插件: Maven提供了 help 插件帮助你了解maven使用的模型，以及某个插件的目标。 能帮你列出活动的profiles 实际的POM(effective POM) ， 打印实际setting (effective settings) 或者列出maven插件的属性, 一般格式为 mvn plugin:goal -Dparameters

mvn help:describe –Dplugin=help –Ddetail=true

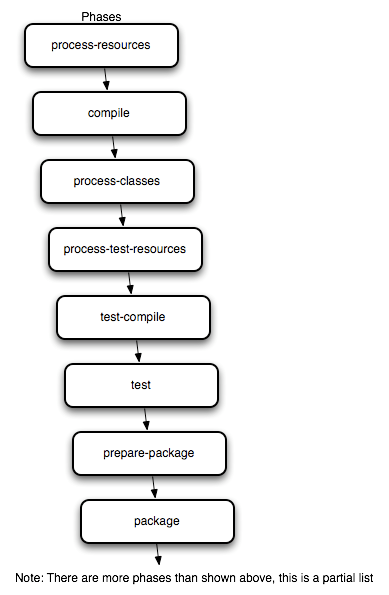
mvn help 插件比较常用的目标有 active-profiles, effective-pom 和 effective-settings 和 describe

* mvn help:describe –Dplugin=help –Dfull
* mvn help:describe –Dplugin=help –Ddetail=true –Dmojo=describe
* mvn help:describe –Dplugin=help –Ddetail=true –Dgoal=describe
* mvn help:describe –Dplugin=help -Dfull –Dgoal=describe

1. 使用 archetype 插件从空白创建简单项目
   1. mvn archetype:generate –DgroupId=org.stanley –DartifactId=simple
2. maven核心概念
   1. plugin: 一个或者多个目标的集合。
   2. goal: 一个明确的任务，可以作为单独的目标运行，也可以作为一个大的构建的一部分和其它目标一起运行。archetype:create 通过plugin:goal 的形式唯一确定 goal



* 1. lifecycle: 对于命令 mvn install 并没有指定一个插件目标，而是指定了一个maven生命周期阶段。一个阶段是被maven称为“构建生命周期”中的一个步骤。生命周期是包含再一个项目构建中的一系列有序的阶段。Maven可以支持很多不同的生命周期，但是最常用的是默认的周期。



插件目标可以附着在生命周期阶段上。随着maven 沿着生命周期的阶段移动，它会执行在特定阶段的目标。 每个阶段可能绑定了零个或者多个目标。

Maven在执行某个阶段时候，首先会有序的执行该阶段之前的所有阶段，到命令行指定的阶段为止。

总结得来说,当我们运行 mvn package, Maven 运行到打包为止的所有阶段,在 Maven 沿着生命周期一步步向前的过程中,它运行绑定在每个阶段上的所有目 标。你也可以像下面这样显式的指定一系列插件目标,以得到同样的结果:

mvn resources:resources compiler:compile resources:testResources compiler:testCompile surefire:test jar:jar

* 1. maven坐标定义了一组标识用来唯一确定一个项目，一个依赖或者maven pom的一个插件。Maven 坐标通常用冒号来作为分隔符来书写,像这样的格式: groupId:artifactId:packaging:version
     1. groupId 公司，团体，组织的团体约定
     2. artifactId 在groupId下标识一个单独项目的唯一标识符
     3. version 项目的特定版本。
     4. packaging 打包格式也是maven的重要组成部分，但不是项目唯一标识符的一部分。默认是jar

在maven化项目构成的巨大空间里，这4个元素是定位使用特定项目的关键元素。

* 1. Maven 仓库是通过结构来定义 的,一个 Maven 仓库是项目构件的一个集合,这些构件存储在一个目录结构下 面,它们的格式能很容易的被 Maven 所理解。在一个 Maven 仓库中,所有的东 西存储在一个与 Maven 项目坐标十分匹配的目录结构中。

Maven 仓库的标准是按照下面的目录格式来存储构件, 相对于仓库的根目录:

**/<groupId>/<artifactId>/<version>/<artifactId>-<version>.<packaging>**

1. Maven 依赖管理

一个复杂的项目将会包含很多依赖,也有可能包含依赖于其它构件的依赖。这是 Maven 最强大的特征之一,它支持了传递性依赖(transitive dependencies)。假如 你的项目依赖于一个库,而这个库又依赖于五个或者十个其它的库(就像 Spring 或者 Hibernate 那样)。你不必找出所有这些依赖然后把它们写在你的 pom.xml 里, 你只需要加上你直接依赖的那些库,Maven 会隐式的把这些库间接依赖的库也加 入到你的项目中。Maven 也会处理这些依赖中的冲突,同时能让你自定义默认行 为,或者排除一些特定的传递性依赖。

1. 由于传递依赖的存在，很多时候我们需要知道项目显式/隐藏的依赖。我们可以通过 dependency 插件获得 获得所有依赖 mvn dependency:resolve

获得依赖树 mvn dependency:tree

1. 多模块项目

一个多模块项目通过一个父POM引用一个或多个子模块来定义。父模块必须指定package类型为POM，在pom.xml中定义子模块。

mvn archetype:generate –DgroupId=org.stanley –DartifactId=parent

change packaging type from jar to pom

Then any new project generated under this parent project will be added as s sub module.

当maven执行一个带有子模块的项目时，maven首先载入父pom，然后定位所有子模块的pom。Maven将所有这些项目的pom放入一个称为 maven 反应堆(reactor) 中，由它解析模块的依赖关系。

1. dependencies/dependencyManagement

Maven 使用dependencyManagement 元素来提供了一种管理依赖版本号的方式。通常会在一个组织或者项目的最顶层的父POM 中看到dependencyManagement 元素。使用pom.xml 中的dependencyManagement 元素能让所有在子项目中引用一个依赖而不用显式的列出版本号。Maven 会沿着父子层次向上走，直到找到一个拥有dependencyManagement 元素的项目，然后它就会使用在这个dependencyManagement 元素中指定的版本号。

dependencyManagement里只是声明依赖，并不实现引入，因此子项目需要显式的声明需要用的依赖。

相对于dependencyManagement，所有声明在dependencies里的依赖都会自动引入，并默认被所有的子项目继承。

1. 版本重复定义

<properties>

<hibernate.version>3.3.0.ga</hibernate.version>

</properties>

1. 兄弟依赖

子项目中，不同项目之间的依赖我们称之为兄弟依赖。可以利用内置的属性${project.groupId}, ${project.version}来解决问题

<dependency>

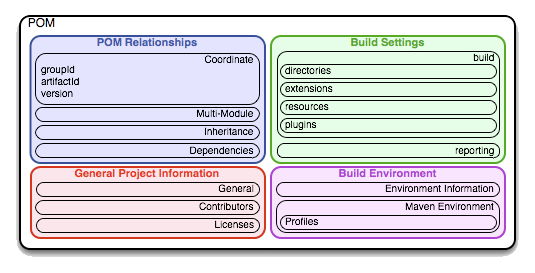
<groupId>${project.groupId}</groupId> <artifactId>simple-weather</artifactId> <version>${project.version}</version>

</dependency>

1. 该元素中配置的依赖插件不会造成实际插件的行为，当子模块 pom 中配置了真正的plugin 元素，

并且 groupId、artifactId 与 pluginManagement 中配置的插件匹配时，pluginManagement 的配置才会影响实际的插件行为。

1. POM对象模型



POM包含了四类描述和配置:

* 项目总体信息： 项目的名称，URL，发起组织等等信息。
* 构建设置： 定义maven构建的默认行为，我们可以更改源代码的位置，添加新的插件，可以将插件目标绑定到生命周期。
* 构建环境：包含了一些能在不同环境中激活的profile。
* POM关系：定义自身的坐标，子模块。

1. 超级POM

所有的maven项目的POM都扩展自超级POM， 它可以在maven 的lib下找到。

1. 简单POM

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>org.sonatype.mavenbook.ch08</groupId>

<artifactId>simplest-project</artifactId>

<version>1</version>

</project>

1. 有效POM

由于POM可以从其它POM继承属性，因此最终获得的有效POM各个POM有效组合。

1. maven项目版本

<major version>.<minor version>.<incremental version>-<qualifier>

限定版本用来标识里程碑构建alpha 和beta发布，限定版本通过连字符与主版本，次版本或增量版本隔离。例如1.3-beta-01 有一个主版本1，次版本3 和限定版本 beta-01.

如果你的版本号与上述格式匹配，它就能正确被匹配。 因此1.2.3 将比1.0.2 更新，这种比较基于主，次，增量版本的数值进行，如果没有符合上述标准，那么你的版本号将只会根据字符串进行比较。

1. 版本限定号

我们还需要对版本号的限定版本进行排序，以版本号1.2.3-alpha-2 和1.2.3-alpha-10为例，虽然alpha-10应该被认为比alpha-2更新，但事实并非如此，问题在于maven对限定版本基于字符串进行比较，因此对于上述比较，我们可以1.2.3-alpha-02 来解决。

1. SNAPSHOT版本

Maven可以包含一个字符串字面量来表示项目正处于活动的开发状态。如果版本包含字符串SNAPSHOT， maven就会在安装发布这个组件时将其展开为一个日期时间值，转换为UTC. 因此你的项目里有个版本 1.0-SNAPSHOT 那么例如在UTC2008年2月7号的下午11:08构建组件时，maven将其展开为1.0-20080207-230803-1. 因此当你发布一个snapshot时，你没有发布一个软件模块，只是发布了一个特定时间的快照版本。

默认设置,Maven 不会从远程仓库检查 SNAPSHOT 版本,要依赖于 SNAPSHOT 版本,用户必须在 POM 中使用 repository 和 pluginRepository 元素 显式的开启下载 snapshot 的功能。

1. 属性引用

一个POM可以通过大括弧和一个美元符号 来包含对属性的引用。

<project>

<groupId>stanley</groupId>

<artifactId>test</artifactId>

<finalName>

${project.groupId}-${project.artifactId}

</finalName>

</project>

maven提供了三个隐式的变量用来访问环境变量，POM信息和maven settings

* env: 暴露了操作系统或者shell的环境变量。${env.PATH}
* project: 暴露了POM，可以通过点标记来引用POM元素的值。
* settings： 暴露了maven setting信息，通过点标记来引用settings.xml中元素点值，例如 ${settings.offline} 来引用settings.xml中offline元素的值。

除了上述隐式的变量， 还可以引用系统属性，以及任何在maven POM中和构建profile中自定义的属性组。

* 所有可以通过　java.lang.System 中 getProperties方法访问的属性都被暴露成POM属性。 例如${user.name}, ${user.home}
* 还可以通过POM.xml 或者 settings.xml 的properties元素设置自己的属性， 或者还可以使用外部载入的文件中属性。如果你在pom.xml设置了一个 fooBar的属性，我们可以通过${fooBar}.

<properties>

<foo>bar</foo>

</properties>

1. 项目依赖范围

* compile 默认的范围， 在所有的classpath中可用，同时也会被打包。
* provided 只在编译classpath可用，不具有传递性，不会被打包。
* runtime 运行和测试系统时需要，编译时候不需要。
* test 在编译，运行时不需要，只在编译测试，运行测试时需要。
* system 与provided类似，但是你必须显式提供一个对于本地系统中jar文件路径。

1. 可选依赖

某些时候你的项目依赖一些组件，但是不需要这些组件被传递到依赖你的项目的组件。 因此你可以将其设置为 optional， 然后在依赖你的项目中显式指定依赖。

1. 依赖版本界限

在指定依赖时，你可以指定一个版本界限。

(, ) 不包含量词

[, ] 包含量词

例如[3.8, 4.0) 意味这 3.8 <= 版本 < 4.0

1. 依赖冲突解决

* 有很多时候你需要排除一个传递性依赖,比如当你依赖于一个项目,后者又继而依赖于 另外一个项目,但你的希望是,要么整个的排除这个传递性依赖,要么用另外一个提供 同样功能的依赖来替代这个传递性依赖

<dependency>

<groupId>org.sonatype.mavenbook</groupId>

<artifactId>project-a</artifactId>

<version>1.0</version>

<exclusions>

<exclusion>

<groupId>org.sonatype.mavenbook</groupId>

<artifactId>project-b</artifactId>

</exclusion>

</exclusions>

</dependency>

1. 依赖管理

当在一个超级复杂的项目中采用maven后，管理版本会成为一个大问题。Maven在 dependencyManagement 元素中为你提供了一种方式来统一管理版本。dependencyManagement 元素能让你在子项目中引用一个依赖而不用显式列出版本号， maven会沿着父子层次往上走，直到找到一个拥有dependencyManagement元素的项目，然后它就会使用在这里指定的版本号。

1. 项目继承

有些情况你会想要一个项目从父POM中继承一些值。可以通过parent 元素使用继承，你就可以避免这种重复。 当一个项目声明parent时候，它从父项目的POM中继承信息，它也可以覆盖父POM的值，或者添加一些新的值。

<project>

<parent>

<groupId>com.training.killerapp</groupId>

<artifactId>a-parent</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<artifactId>project-a</artifactId>

</project>

Maven 假设父 POM 在本地仓库中可用,或者在当前项目的父目录(../pom.xml) 中可用。

如果两个位置都不可用,默认行为还可以通过 relativePath 元素被覆盖。

1. 生命周期

maven 生命周期包含了一些有序的命名阶段， 有些阶段抽象了编译过程，有些抽象了打包过程。

Maven 中有三种标准的生命周期: clean, default, site 。

1. clean 生命周期

mvn clean将调用清理生命周期，包含三个阶段

* pre-clean
* clean
* post-clean

1. 默认生命周期

默认周期是应用程序构建过程的总体模型。

1. 站点生命周期

Maven 不仅仅能从一个项目构建软件构件,它还能为一个或者一组项目生成项目文档

和报告。项目文档和站点生成有一个专有的生命周期

1. 打包相关生命周期

绑定到每个阶段的特定目标默认根据项目的打包类型设置，一个打包类型为jar的项目和war的项目拥有不同的默认目标。

1. 构建profile 能让你为一个特殊的环境定义一个特殊的构建，profile是的不同环境间构建的可移植性成为可能。

<profiles>

<profile>

<id>production</id>

<build>

<plugins>

<plugin>

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

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

<configuration>

<debug>false</debug>

<optimize>true</optimize>

</configuration>

</plugin>

</plugins>

</build>

</profile>

</profiles>

* POM.xml 中的profiles元素包含一个或多个 profile元素，由于它会覆盖pom.xml中的默认设置，因此通常为最后一个元素。
* 每个profile必须有一个id， 调用时通过 –P<profile\_id> 调用 profile

1. 外部profile

如果你开始大量使用 Maven profile,你会希望将 profile 从 POM 中分离,使用一个 单独的文件如 profiles.xml。你可以混合使用定义在 pom.xml 中和外部 profiles.xml 文件中的 profile。只需要将 profiles 元素放到${basedir}目录下的 profiles.xml 文件中,然后照常运行 Maven 就可以。

1. Settings Profile

项目 profile 关心于覆盖 某个项目的配置,而 settings profile 可以应用到所有你使用 Maven 构建的项目

1. mvn help:active-profiles 可以列出当前所有活动的 profiles
2. dd
3. d
4. d
5. d
6. d
7. d
8. d
9. d