

并发编程网 - ifeve.com

让天下没有难学的技术

[HOME](#)[JAVA](#)[《Maven官方指南》30分钟入门](#)

SEARCH



JUN
201704

1,534 人阅读

魔术师Carvendy

JAVA

☆☆☆☆ (2
votes, average: **2.00**
out of 5)

Write comment

《Maven官方指南》30分钟入门

[原文链接](#) 译者:carvendy

Maven 入门指南

这个指南预期是作为使用第一次使用Maven工作的，但是也预期服务作为一个单独的引用和常用的解决方案用例。对第一次的用户，这推荐你按顺序一步一步来看材料。对于熟悉Maven的用户，指南努力地提供需要的快速方案。假设你已经下载了Maven和安装好Maven在你本地机器。如果你这样做请参考[下载与安装](#)的指令。

好，你现在已经安装好Maven了和我们准备开始。在我们今天例子之前，我们将简要过一遍什么是Maven和它能怎么帮助你处理日常工作并和你团队成员一起协助。Maven将，当然，为了小项目工作，但是Maven表现突出在帮助团队有效操作并允许团队成员着眼于项目需要什么相关者。你可以把构建的事项交给Maven。

选项

- [什么是Maven ?](#)
- [使用Maven对我的开发进程有怎么样好处 ?](#)
- [怎么安装Maven ?](#)
- [怎么创建我第一个Maven项目 ?](#)
- [怎么编译我的项目源码 ?](#)
- [怎么编译我的测试代码和运行我的单元测试 ?](#)
- [怎么创建一个JAR和安装它到我本地仓库 ?](#)
- [什么是快照版本 ?](#)
- [怎么使用插件 ?](#)
- [怎么加配置到我的JAR ?](#)



RECENT POSTS

- [《RabbitMQ官方指南》主题](#)
- [《Nginx官方文档》关于nginxScript](#)
- [《RabbitMQ官方指南》路由](#)
- [《RabbitMQ官方指南》RPC](#)
- [《Nginx官方文档》用DTrace pid提供程序调试nginx](#)
- [《Nginx官方文档》转换重写规则](#)
- [《Nginx官方文档》WebSocket代理](#)

- [怎么过滤资源文件？](#)
- [怎么使用额外的依赖？](#)
- [怎么打包我的jar到我的远程仓库？](#)
- [我怎么创建文档？](#)
- [我怎么构建其他类型的项目？](#)
- [我怎么一次构建超过一个项目？](#)

什么是Maven？

在一眨眼看Maven有很多东西，但是在简单的来说Maven中在尝试使用规则对一个项目的构建基础设施，为了促进理解和生产力提供了一个明确的路径在最佳实践中。Maven基本是一个项目管理和理解的工具就提供一种方式让我们管理：

- 构建
- 文档
- 报表
- 依赖
- SCMs
- 发布
- 分布式

如果你想更多Maven背后的信息你可以看看[Maven的哲学](#)和Maven的历史。现在让我们转移话题到，用户能从Maven中获取什么好处。

使用Maven对我的开发进程有怎么样好处？

Maven可以提供好处让你在构建进程中采用标准交流和在实践中加快你的开发周期与此同时帮助你得到一个高效率的成功。 现在我们有覆盖一些小历史和Maven的目的，现在让我们进入一些真实的例子让你运行Maven！

怎么安装Maven？

Maven的默认设置通常是足够的，但如果你需要更改缓存位置或在HTTP代理，你将需要创建配置。看[Maven配置指南](#)里面有更多信息。

怎么创建我第一个Maven项目？

我们要一头扎进创建你的第一个Maven工程！要创建我们的第一个Maven项目我们将使用Maven的原型机制。原型被定义为一个原始的模式或模型，从所有其他同类的东西。在Maven，原型是一个工程模板，并结合用户输入产生一个Maven项目已经根据用户的要求。现在我们将展示给你怎么使用模板机制工作，但是如果你想了解更多关于模板请看我们的[模板介绍](#)。

关于创建您的第一个项目！为了创建简单的Maven项目，从命令行执行以下：

```
mvn -B archetype:generate \
    -DarchetypeGroupId=org.apache.maven.archetypes \
```

- [《RabbitMQ官方指南》翻译邀请](#)
- [JAVA8 stream 中Spliterator的使用\(二\)](#)
- [JAVA8 stream 中Spliterator的使用\(一\)](#)
- [Spring Boot 整合 Thymeleaf 完整 Web 案例](#)
- [实战Spring事务传播性与隔离性](#)
- [nginx如何处理TCP / UDP会话](#)
- [《Nginx官方文档》配置HTTPS服务器](#)
- [《TensorFlow官方文档》快速入门](#)
- [《Nginx官方文档》Nginx之Server names](#)
- [Spring事务配置解惑](#)
- [《Spring 5 官方文档》39. 创建可扩展的XML](#)
- [实战解析—论三年内快速成长为一名技术专家](#)
- [软件架构模式-第二章事件驱动架构\(上\)](#)
- [《阿里感悟》如何在三年内成长为一名技术专家](#)
- [软件架构模式-第一章分层架构（下）](#)
- [《Maven官方文档》配置默认Mojo扩展](#)
- [《Hibernate快速开始 – 4 – 使用JAVA持久层 API \(JPA\) 教程》](#)
- [《Hibernate快速开始》翻译邀请](#)
- [Java面试题-基础知识](#)
- [JDK8中新增原子性操作类LongAdder](#)
- [《Maven官方文档》Maven 文档风格指南](#)
- [《Maven官方文档》HttpClient HTTP提供者的先进配置](#)
- [应届生JAVA技术面试题](#)

```
-DgroupId=com.mycompany.app \
-DartifactId=my-app
```

一旦你执行了这个命令，你会注意到一些事情发生了。首先，你会注意到一个目录命名为**my-app**已经为新项目创建的，这个目录包含一个文件名为pom.xml，看起来应该是这样的：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>

  <artifactId>my-app</artifactId>

  <packaging>jar</packaging>

  <version>1.0-SNAPSHOT</version>

  <name>Maven Quick Start Archetype</name>

  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.11</version>

      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

pom.xml 包含项目的项目对象模型。这个POM是Maven最基本的工作单元。重要的是记住因为Maven是固有的项目式环绕这项目的概念。简单的说，POM包含了关于项目的每一个重要信息，基本上是一站式的，可以找到与项目相关的任何东西。理解POM很重要和鼓励新用户参考[POM介绍](#)。

这是一个非常简单的POM，但仍然显示每个POM包含的关键要素，所以让我们走过每一个熟悉你的POM要领：

project 这是Maven pom.xml文件中最顶级的节点。

modelVersion 这个节点指示 POM的对象模型试验什么版本。模型的版本改变非常少但是这是强制性的为了确定使用的稳定性和当Maven开发者认为至为重要的是改变这个模型。

热门文章

[Google Guava官方教程（中文版）](#) 516,132 人阅读

[Java NIO系列教程（一）Java NIO 概述](#) 333,142 人阅读

[Java并发性和多线程介绍目录](#) 227,368 人阅读

[Java NIO 系列教程](#) 221,428 人阅读

[Java NIO系列教程（十二）Java NIO与IO](#) 189,945 人阅读

[Java NIO系列教程（六）Selector](#) 170,576 人阅读

[Java NIO系列教程（二）Channel](#) 165,333 人阅读

[Java NIO系列教程（三）Buffer](#) 164,641 人阅读

[《Storm入门》中文版](#) 163,143 人阅读

[Java8初体验（二）Stream语法详解](#) 162,213 人阅读

[69道Spring面试题和答案](#) 145,950 人阅读

[Netty 5用户指南](#) 135,333 人阅读

[Java 7 并发编程指南中文版](#) 126,975 人阅读

[并发框架Disruptor译文](#) 121,543 人阅读

[Java NIO系列教程（八）SocketChannel](#) 110,338 人阅读

[\[Google Guava\] 2.3-强大的集合工具类：ja...](#) 100,353 人阅读

[\[Google Guava\] 3-缓存](#) 99,893 人阅读

[\[Google Guava\] 1.1-使用和避免null](#) 99,310 人阅读

[Java NIO系列教程（七）FileChannel](#) 98,295 人阅读

groupId 这个节点指示是创建项目的组织或组的唯一标识符。这个groupId是项目键标识符的其中一个和通常基于完整有资格的组织领域名字。例子：`org.apache.maven.plugins` 是设计的一个组对于所有Maven插件。

artifactId 这个节点指示是私有工件本被项目生成的唯一基本名字。这个项目的私有工件通常是一个JAR文件。第二，工件像源码捆也用于这个工件作为他们最终的名字。一个通常的由Maven生成的工件会有表单<artifactId>.<version>.<extension>（例如 `myapp-1.0.jar`）。

packaging 这个节点是用于这个工件肺打包类型(例如 JAR,WAR,EAR 等等)。这不仅意味着工件生成的是JAR,WAR,EAR但是指示是一个指定生命周期使用，作为构建进程的一部分。（这个周期是一个话题我们将处理并进一步研究在这个指南里。对于现在，只要记住项目的packaging指示可以作为一部分在自定义构建周期中。）**packaging**的默认值是JAR 所以你不需为大部分项目指定。

version 这个节点指示着这个工件被创建的版本。Maven 有很长的一段路来帮助你管理版本和你经常看的**SNAPSHOT**指示器在这个版本中，它表示这这个项目是正在开发的状态。我们将讨论快照的[使用](#)和他们怎么进一步使用在指南中。

name 这个节点指示这项目显示名字,这是经常使用Maven的生成文件。

url 此元素指示项目的网站可在何处找到。这是经常使用Maven的生成文件。

description 此元素为您的项目提供基本说明。这是经常使用Maven的生成文件。

在您的第一个项目的原型生成之后，您还将注意到以下目录结构已被创建：

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    |-- test
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- AppTest.java
```

正如你所看到你，项目创建来源于模板有一个POM，一个源码树是你程序源码和一个源码树是你测试的源码。这个是Maven项目标准的结构（程序源码放在 `${basedir}/src/main/java` 和测试代码在`${basedir}/src/test/java`，`${basedir}`代表目录包含在 `pom.xml`）

如果你手动创建一个Maven项目使用的目录结构要是我们推荐使用的。这是Maven的沟通方式和学习更多你可以读我们[标准文档结构](#)。

现在我们有一个POM,一些项目源码和一些测试源码，你可能会问。。。

TAGS

actor Basic book classes collections
concurrency Concurrent concurrent
data structure Customizing Executor
Executor framework False Sharing faq fork
Fork/Join fork join Framework Functional
Programming Guava IO **JAVA** java8
jmm join JVM lock Memory Barriers Netty
NIO OAuth 2.0 pattern-matching RingBuffer Scala
slf4j spark spark官方文档 stm Storm
synchronization Synchronized thread
tomcat volatile 多线程 **并发译文，Java**
，Maven

怎么编译我的项目源码？

改变目录在pom.xml的 archetype:generate和执行，通过下面的命令编译你程序源码：

```
mvn compile
```

在执行这个命令的时候你可以看到输出如下：

```
[INFO] -----
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [compile]
[INFO] -----
[INFO] artifact org.apache.maven.plugins:maven-resources-plugin: \
    checking for updates from central
...
[INFO] artifact org.apache.maven.plugins:maven-compiler-plugin: \
    checking for updates from central
...
[INFO] [resources:resources]
...
[INFO] [compiler:compile]
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 minutes 54 seconds
[INFO] Finished at: Fri Sep 23 15:48:34 GMT-05:00 2005
[INFO] Final Memory: 2M/6M
[INFO] -----
```

第一次执行这个命令（或其他命令）的时候，Maven需要下载它需要完成命令的所有插件和相关依赖。从一个Maven的清晰安装，这可能需要相当长的时间（在上面输出，可能要4分钟）。如果你再次执行这个命令，Maven现在所需要的时间变短了，它将不会下载任何新的和将能执行这个命令相当快。

正如你在输出能看到的，编译class的地方在 `${basedir}/target/classes`，那里是另一个标准的交流被Maven使用的。所以，如果你是一个热心的观察者，你将会注意到使用标准的交流方式，POM上是非常小和你将不用明确地告诉Maven你任何源码在哪里和应该在哪里输出。通过Maven的标准交流方式，你可以做得更少了！比较休闲，让我们看看你可能要做的在[Ant](#)要完成的同样事情。

ARCHIVES

[October 2017](#) (16)

[September 2017](#) (10)

[August 2017](#) (25)

[July 2017](#) (20)

[June 2017](#) (38)

[May 2017](#) (59)

[April 2017](#) (17)

[March 2017](#) (30)

[February 2017](#) (9)

[January 2017](#) (7)

[December 2016](#) (12)

[November 2016](#) (27)

[October 2016](#) (16)

[September 2016](#) (11)

[August 2016](#) (6)

[July 2016](#) (9)

[June 2016](#) (7)

[May 2016](#) (20)

[April 2016](#) (28)

[March 2016](#) (8)

[February 2016](#) (7)

友情链接

[coolshell](#)

[Programer. 大猫](#)

[一粟的博客](#)

[志俊的博客](#)

[最代码](#)

[点点折](#)

[领悟书生](#)

CATEGORIES

[akka](#) (20)

[Android](#) (3)

[C++](#) (12)

[CPU](#) (2)

[Framework](#) (54)

[GO](#) (6)

[groovy](#) (6)

[guava](#) (23)

[JAVA](#) (724)

[JVM](#) (37)

[linux](#) (8)

怎么编译我的测试代码和运行我的单元测试？

现在您已经成功编译应用程序的源代码，现在已经有一些单元测试要编译和执行了(因为没程序总是写和执行他们单元测试 *轻推眨眼眨眼*)

执行下面的命令

```
mvn test
```

在执行这个命令的时候你应该会看到输出下面的：

```
[INFO] -----
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [test]
[INFO] -----
[INFO] artifact org.apache.maven.plugins:maven-surefire-plugin: \
    checking for updates from central
...
[INFO] [resources:resources]
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] [compiler:testCompile]
Compiling 1 source file to C:\Test\Maven2\test\my-app\target\test-classes
...
[INFO] [surefire:test]
[INFO] Setting reports dir: C:\Test\Maven2\test\my-app\target\surefire-reports

-----

T E S T S

-----

[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0 sec

Results :

[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] -----
[INFO] BUILD SUCCESSFUL
```

January 2016 (10)	Netty (31)
December 2015 (15)	react (6)
November 2015 (24)	redis (22)
October 2015 (17)	Scala (11)
September 2015 (29)	spark (19)
August 2015 (44)	Spring (18)
July 2015 (20)	storm (44)
June 2015 (21)	thinking (3)
May 2015 (15)	Velocity (10)
April 2015 (27)	Web (18)
March 2015 (13)	zookeeper (1)
February 2015 (11)	公告 (5)
January 2015 (13)	大数据 (33)
December 2014 (26)	好文推荐 (31)
November 2014 (61)	并发书籍 (96)
October 2014 (33)	并发译文 (398)
September 2014 (47)	感悟 (3)
August 2014 (27)	技术问答 (12)
July 2014 (13)	敏捷管理 (6)
June 2014 (32)	本站原创 (86)
May 2014 (45)	架构 (29)
April 2014 (41)	活动 (6)
March 2014 (34)	网络 (6)
February 2014 (38)	
January 2014 (19)	
December 2013 (10)	
November 2013 (4)	
October 2013 (20)	
September 2013 (38)	
August 2013 (48)	
July 2013 (26)	
June 2013 (16)	
May 2013 (9)	
April 2013 (17)	


```
[INFO] -----  
[INFO] Total time: 15 seconds  
[INFO] Finished at: Thu Oct 06 08:12:17 MDT 2005  
[INFO] Final Memory: 2M/8M  
[INFO] -----
```

关于这个输出一些要注意的事情

Maven这时下载了很多依赖。这些依赖和插件对执行单元测试是很重要的（它可能已经有了这些它需要的依赖那么将不会再下载）。在正在编译和执行单元测试之前 Maven会 编译主要的代码（所有class是到目前为止因为它们不会再改变任何东西在最新的编译中）

如果你想简单编译你的测试代码（但是不执行测试代码），你可以执行：

```
mvn test-compile
```

现在，您可以编译应用程序源，编译测试，并执行测试，您将继续进行下一个逻辑步骤，这样您就可以询问了...

怎么创建一个JAR和安装它到我本地仓库？

下面的命令执行成功之后，会生成一个jar文件：

```
mvn package
```

如果你看看你的项目POM你将会注意到 **packaging** 节点设置的是**jar**。这就是Maven怎么知道从上面的命令生成一个JAR（我们将讨论这在后面）。你可以看看 `${basedir}/target`目录和你将看到生成了一个JAR文件。

现在你将要安装你生成的工件（jar file）在你本地仓库（默认是在 `${user.home}/.m2/repository`）。想要获取很多关于仓库的信息你可以看我们的[仓库的介绍](#)但是让我们转移到安装我们的工件！执行下面的指令：

```
mvn install
```

执行此命令后，你应该看到以下输出：

```
[INFO] -----  
[INFO] Building Maven Quick Start Archetype  
[INFO]    task-segment: [install]  
[INFO] -----  
[INFO] [resources:resources]  
[INFO] [compiler:compile]
```

[March 2013](#) (41)

[February 2013](#) (25)

[January 2013](#) (57)

[December 2012](#) (9)

[October 2012](#) (1)

[August 2012](#) (1)



```
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] [resources:testResources]
[INFO] [compiler:testCompile]
Compiling 1 source file to <dir>/my-app/target/test-classes
[INFO] [surefire:test]
[INFO] Setting reports dir: <dir>/my-app/target/surefire-reports

-----

T E S T S

-----

[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.001 sec

Results :
[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] [jar:jar]
[INFO] Building jar: <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar to \
    <local-repository>/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 5 seconds
[INFO] Finished at: Tue Oct 04 13:20:32 GMT-05:00 2005
[INFO] Final Memory: 3M/8M
[INFO] -----
```

请注意，正确的插件（进行测试）是包含在文件与特定的命名约定试验。默认测试包括：

- **/*Test.java**
- */Test.java**
- **/*TestCase.java**

和默认不包含：

- */AbstractTest.java**

`*/AbstractTestCase.java`

你已经走过了建立、建设过程、检验、包装、安装一个典型的Maven项目。这可能是绝大多数项目将要使用Maven，你是否注意到，你已经能够做到到了这一点，一切都是由一个18行的文件驱动，即工程模型或POM。如果你看看一个通常的Ant[构建文件](#)它提供了一样的功能，我们已经到达到目前为止你将两次注意到POM的大小和我们正在入门！这对于你是很多可用功能在Maven不需要增加任何东西对于现有的我们POM。为了从我们Ant例子中得到更多功能你必须保持不断犯错误。

所以你能轻松地得到？这有很多Maven插件在盒子里面是一个简单的POM就像上面所看到的。万能将注意到这里特别这是其中一个高价值Maven特性：没有任何工作部分的POM就有足够生成一个web站点对你的项目！你将会喜欢自定义你的Maven site但是你缺少时间，对你需要做的事情而提供的关于项目基本信息是执行下面的指令：

```
mvn site
```

有许多其他的独立目标也可以执行，例如：

```
mvn clean
```

这将在开始之前将所有生成数据移除target目录，以便它是新的。也许你想IntelliJ理念产生的描述符的项目，是吗？

```
mvn idea:idea
```

这可以运行在以前的想法项目的顶部-它会更新设置，而不是开始就是新的。如果你是Eclipse IDE，执行：

```
mvn eclipse:eclipse
```

笔记：一些相似的指令从Maven 1.0保留到现在——就像 `jar:jar`，但是它们可能没有像你希望的习惯。目前，`jar:jar`将不能重新编译源码——这将建议地从target/classes目录 创建JAR，在假设一切已经完成。、

什么是快照版本？

注意version节点的值在pom.xml文件显示有后缀 -SNAPSHOT。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ...
  <groupId>...</groupId>
  <artifactId>my-app</artifactId>
  ...
```

```
<version>1.0-SNAPSHOT</version>

<name>Maven Quick Start Archetype</name>

...

```

快照值是指沿着开发分支的“最新”代码，并不能保证代码是稳定的或不变的。相反，“发布”版本中的代码（没有后缀快照的任何版本值）是不变的。

换句话说，快照版本是最终版本之前的“开发”版本。快照是比发行版老。

在释放过程中，一个版本的变化 $x.y\text{-SNAPSHOT} (Y + 1)$ -快照。例如，1.0-SNAPSHOT，以及新的开发版是1.1-SNAPSHOT。

怎么使用插件？

每当你想要定制Maven项目的建设，这是通过增加或重新配置插件做的。**为Maven 1.0用户的笔记：**在1.0中，你将加入一些 **preGoal**到 **maven.xml**文件中和一些实体到**project.properties**.这里有一点不同在这个例子，我们将配置运行使用JDK 5.0编译编译源码。这是简单添加到你的POM:

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.3</version>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
...

```

你会发现所有的Maven插件看起来非常像一个依赖，在某些方面，他们是。此插件将自动下载和使用-包括一个特定的版本，如果你要求它（默认是使用最新可用）。

configuration元素允许从编译插件中给定参数对每一个指令。在上面的用例，编译插件已经作为构建进程一部分而使用和改变配置。这也能添加新的指令在进程，和配置具体的指令。需要跟多信息，看看[构建生命周期介绍](#)。

为了找到配置什么对于一个插件是可用的，你可以看看[插件列表](#)并引导插件和指令的使用。为了常用的信息关于怎么配置可用的插件参数，你看看[配置插件指南](#)。

怎么加配置到我的JAR ？

另外常用的用例可以满足不修改POM,我们上面有包装资源的jar文件。对于常用的任务，Maven可以再次依赖[标准目录结构](#)，这意味着使用标准的Maven交流方式你可以使用JAR打包资源并容易放置资源在标准的文件目录。

你可以在下面的例子，我们添加了目录 `${basedir}/src/main/resources` 并放置到我们希望打包的JAR中。这是被Maven使用的规则：任何目录和文件放置在 `${basedir}/src/main/resources` 目录就会被打包到你的JAR,确切相同的结构在JAR的开始目录。

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- com
    |   |       `-- mycompany
    |   |           `-- app
    |   |               `-- App.java
    |   `-- resources
    |       `-- META-INF
    |           `-- application.properties
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

所以你可以看到，在我们的例子中，我们有一个application.properties META-INF目录，目录内文件。如果你打开JAR，Maven为您创建了看它，你会看到下面的：

```
|-- META-INF
|   |-- MANIFEST.MF
|   |-- application.properties
|   `-- maven
```

```
|      |-- com.mycompany.app
|
|      |-- my-app
|
|          |-- pom.properties
|
|          |-- pom.xml
|-- com
|
|    |-- mycompany
|
|        |-- app
|
|            |-- App.class
```

正如你所看到的，`${basedir}/src/main/resources`的内容可以在JAR的开始目录被找到 和我们的`application.properties` 文件在 `META-INF`目录 你将需要注意其他文件就像 `META-INF/MANIFEST.MF`和`pom.xml`和`pom.properties`文件差不多。这是Maven生成的标准JAR.你可以创建自己的manifest如果你选择的话，但是Maven将会生成一个如果你这么做的话。（ 你可以修改默认mainifest的实体。我们后面再讨论 ）. `pom.xml` 和 `pom.properties`文件被打包到JAR所以每一个被Mave创建的工件是自我描述的和允许你利用在你项目中的元数据如果你需要。在简单的使用可能检索你程序的版本。操作POM文件将需要你使用一些Maven用途但是这配置能被利用标准的Java API和看上去像这样：

```
#Generated by Maven

#Tue Oct 04 15:43:21 GMT-05:00 2005

version=1.0-SNAPSHOT

groupId=com.mycompany.app

artifactId=my-app
```

添加资源到你的单元测试中，你遵循相同的模式作为你的资源添加到罐子里除了你的目录的地方资源为`${basedir}/src/test/resources`。在这一点上，您将有一个项目目录结构，看起来如下：

```
my-app
|-- pom.xml
|-- src
|   |-- main
|   |   |-- java
|   |   |   |-- com
|   |   |       |-- mycompany
|   |   |           |-- app
|   |   |               |-- App.java
|   |-- resources
|   |-- META-INF
|       |-- application.properties
```

```

    |-- test
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   |-- AppTest.java
    |-- resources
    |   |-- test.properties
```

在单元测试中，您可以使用如下代码来访问测试所需的资源：

```
...

// Retrieve resource
InputStream is = getClass().getResourceAsStream( "/test.properties" );

// Do something with the resource

...
```

怎么过滤资源文件？

有时候一个资源文件需要包含一个值可能是构建的时间。为在Maven中了完成它，放置一个引用属性将会包含这个值到你的配置文件使用语法

`${<property name>}`。这个属性可以被定义到你的pom.xml，或定义到你的 settings.xml，或定义到一个额外的配置文件，或者系统属性。

当拷贝的时候为了过滤资源，简易地设置 **filtering**来确定真实目录在你的pom.xml：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```
<name>Maven Quick Start Archetype</name>

<url>http://maven.apache.org</url>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
</project>
```

你将注意到我们不得不添加 **build**, **resources**和 **resource**节点当这里没有 的时候。此外，我们不得不确定放置在src/main/resources目录的资源的状态。以前所有信息提供作为默认值，但是因为默认值对于 **filtering**是false，我们不得不添加到pom.xml以覆盖默认值和设置**filtering**为true.

对于定义一个属性在你的pom.xml，这个属性名字使用xml定义元素，“pom”被允许作为别名对于项目（root）节点。所以 **`${project.name}`** 引用项目的名字，**`${project.version}`**引用项目的版本，**`${project.version.finalName}`** 引用项目打包等构建之后的文件创建名字。笔记：POM的一些元素有默认值，所以不需要确切定义到你的 pom.xml这些值在这里已经够用了。简单地说，这些值在用户的**settings.xml**能被引用使用属性以“settings”开头（例如：**`==${settings.localRepository}==`**引用用户的本地仓库路径）

继续我们的例子，让我们加入一对属性到 **application.properties**文件（我们放在src/main/resources，目录）那些值将会被支持在资源被过滤的时候。

```
# application.properties
application.name=${project.name}
```

```
application.version=${project.version}
```

在这里，你可以执行下面的命令（进程资源是在构建周期里的资源会被拷贝和过滤）

```
mvn process-resources
```

和在 **application.properties**文件在 **target/classes**（最后到达jar里面）像这样：

```
# application.properties  
application.name=Maven Quick Start Archetype  
application.version=1.0-SNAPSHOT
```

为了引用一个属性定义额外的文件，你需要做的是在你的pom.xml中加一个引用到额外的文件。首先，让我们创建我们额外的配置文件我们叫它 **src/main/filters/filter.properties**：

```
# filter.properties  
my.filter.value=hello!
```

接着，我们在pom.xml中加入一个引用到新文件。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  
  <groupId>com.mycompany.app</groupId>  
  <artifactId>my-app</artifactId>  
  <version>1.0-SNAPSHOT</version>  
  <packaging>jar</packaging>  
  
  <name>Maven Quick Start Archetype</name>  
  <url>http://maven.apache.org</url>  
  
  <dependencies>  
    <dependency>  
      <groupId>junit</groupId>
```



```
        <artifactId>junit</artifactId>

        <version>4.11</version>

        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <filters>
        <filter>src/main/filters/filter.properties</filter>
    </filters>
    <resources>
        <resource>
            <directory>src/main/resources</directory>

            <filtering>true</filtering>
        </resource>
    </resources>
</build>
</project>
```

然后，如果我们要在`application.properties`中加入这个属性：

```
# application.properties
application.name=${project.name}
application.version=${project.version}
message=${my.filter.value}
```

然后执行`mvn process-resources`命令将会把新的属性值设置到`application.properties`。作为一个可替代的定义`my.filter.value`属性值在一个额外的文件，你也可以定义在你的`pom.xml`的`properties`部分和你会得到同样的效果（注意我不需要引用`src/main/filters/filter.properties`了）

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
```

```
<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Maven Quick Start Archetype</name>
<url>http://maven.apache.org</url>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>

<properties>
  <my.filter.value>hello</my.filter.value>
</properties>
</project>
```

过滤资源可以让你从系统配置文件里获取值；系统属性构建进Java（像是java.version或user.home）或者属性文件使用命令定义，标准的Java -D参数。为了继续例子，让我们改变一下application.properties 文件像这样：

```
# application.properties
java.version=${java.version}
command.line.prop=${command.line.prop}
```

现在，当你执行下列命令（笔记定义通过命令行定义command.line.prop属性），application.properties文件将包含值通过系统属性获取。

```
mvn process-resources "-Dcommand.line.prop=hello again"
```

怎么使用额外的依赖

你可以已经注意到了在POM中的**dependencies**元素在前面的例子我们已经使用过了。在事实上，你已经使用过额外的依赖了，但是我能将讨论它工作的更多细节。想知道更多介绍，请你参考[依赖机制介绍](#)。

在pom.xml的**dependencies**部分列了所有额外的依赖在我们项目构建需要的（任何需要的依赖在编译，测试，运行，任何时候）。现在，我们的项目只有依赖JUnit（我移除资源过滤的东西为了清晰）

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

对于每一个额外的依赖。你将需要定义至少四个东西：groupId,artifactId,version,scope。在构建依赖的项目pom.xml中也一样有

groupId,artifactId,version。这个scope元素是你的项目怎么样使用依赖，它的值是**compile**,**test**,**run**。为了更多信息你可以知道一个依赖，看看[项目描述引用](#)。

为了更多信息关于依赖机制，看[依赖机制介绍](#)。

关于一个依赖的信息，Maven将不能引用依赖当项目构建的时候。Maven的引用依赖从哪里来？Maven像是在你本地仓库

(`${user.home}/.m2/repository`是默认地址) 来找到所有依赖。在[前面的部分](#)，我们从自己项目安装工件 (`my-app-1.0-SNAPSHOT.jar`) 在本地仓库。安装在这里了，其他项目就可以引用这个jar简单地在pom.xml里面加上依赖信息：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-other-app</artifactId>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>com.mycompany.app</groupId>
      <artifactId>my-app</artifactId>
      <version>1.0-SNAPSHOT</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

依赖了什么构建在其他地方的？怎么得到在本地仓库？无论如何一个项目引用了一个本地仓库没有的依赖，Maven就会通过远程仓库下载到本地仓库.你可以注意到了Maven下载了很多东西在你构建第一个项目的时候（那些下载的依赖为了各种构建项目使用的插件）。默认地，远程仓库可以找到(和被浏览)在<http://repo.maven.apache.org/maven2/>。你可以安装到你自己的远程仓库（可能你是你妈公司的中心仓库）来代替默认的远程仓库。为了获取更多关于仓库的信息你可以参考[仓库的介绍](#)。

让我们添加另一个依赖于我们的项目。让我们说，我们已经添加了一些日志的代码，需要添加log4j作为一个依赖。首先，我们需要知道的GroupID、artifactId，和版本是log4j。我们可以浏览ibiblio寻找它，或用谷歌搜索“网站有所帮助：[www.ibiblio.org maven2 Log4J](http://www.ibiblio.org/maven2/Log4J/)”。搜索显示一个目录名/maven2/log4j/log4j（或 /pub/packages/maven2/log4j/log4j）。这个目录是一个文件名为maven-metadata.xml。这里就是Log4j maven-metadata.xml看起来像：

```
<metadata>
  <groupId>log4j</groupId>
```

```
<artifactId>log4j</artifactId>

<version>1.1.3</version>

<versioning>
  <versions>
    <version>1.1.3</version>
    <version>1.2.4</version>
    <version>1.2.5</version>
    <version>1.2.6</version>
    <version>1.2.7</version>
    <version>1.2.8</version>
    <version>1.2.11</version>
    <version>1.2.9</version>
    <version>1.2.12</version>
  </versions>
</versioning>
</metadata>
```

从这个文件中，我们可以看到，我们要的是“log4j groupId和artifactId是“log4j”。我们看到很多不同版本的价值选择；现在，我们只使用最新的版本，1.2.12（一些maven-metadata.xml文件还可以指定哪个版本是目前发布的版本）。在maven-metadata.xml文件，我们可以看到每一个版本的log4j库对应一个目录。在每一个这些，我们会发现实际的JAR文件（如log4j-1.2.12.jar）以及POM文件（这是依赖，pom.xml表示它可能和其他信息的任何进一步的依赖）和另一个maven-metadata.xml文件。也有一个对应于每个MD5文件，其中包含这些文件的MD5散列。您可以使用此方法来验证该库或找出某个版本的特定的库，您可能已经使用。

现在我们知道我们需要的信息，我们可以依赖我们的pom.xml文件：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```
<name>Maven Quick Start Archetype</name>

<url>http://maven.apache.org</url>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.12</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
</project>
```

现在，当我们编译项目(mvn compile)。我们将看看Maven为我们下载的log4j的依赖。

怎么发布我的jar到我的远程仓库？

部署jar外部储存库，您必须在pom.xml配置库的URL和连接在settings.xml库认证信息。 这里是一个例子，使用SCP和用户名/密码身份验证：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.codehaus.plexus</groupId>
    <artifactId>plexus-utils</artifactId>
    <version>1.0.4</version>
  </dependency>
</dependencies>

<build>
  <filters>
    <filter>src/main/filters/filters.properties</filter>
  </filters>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
<!--
|
|
|
-->
<distributionManagement>
  <repository>
    <id>mycompany-repository</id>
    <name>MyCompany Repository</name>
    <url>scp://repository.mycompany.com/repository/maven2</url>
  </repository>
```



```
</distributionManagement>

</project>
```

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">

  ...

  <servers>

    <server>

      <id>mycompany-repository</id>

      <username>jvanzyl</username>

      <!-- Default value is ~/.ssh/id_dsa -->

      <privateKey>/path/to/identity</privateKey> (default is ~/.ssh/id_dsa)

      <passphrase>my_key_passphrase</passphrase>

    </server>

  </servers>

  ...

</settings>
```

注意，如果你是连接到一个OpenSSH SSH服务器具有参数“密码验证”设置为“无”的sshd_config，你必须输入你的密码每次用户名/密码认证（尽管你可以登录使用一个SSH客户端通过键入用户名和密码）。在这种情况下，您可能需要切换到公钥验证。

应该在settings.xml设置如果你需要使用密码的时候。为了了解更多信息，看看[密码加密](#)。

怎么创建文档？

让你开始使用Maven的文件系统，你可以使用原型机制来生成一个网站的现有项目中使用以下命令：

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-site \
  -DgroupId=com.mycompany.app \
  -DartifactId=my-app-site
```

现在头上有[创建网站指南](#)可以学习一些你的项目怎么样创建文档。

怎么构建其他类型的项目？

注意，生命周期适用于任何项目类型。例如，回到基本目录中，我们可以创建一个简单的web应用程序：

```
mvn archetype:generate \
    -DarchetypeGroupId=org.apache.maven.archetypes \
    -DarchetypeArtifactId=maven-archetype-webapp \
    -DgroupId=com.mycompany.app \
    -DartifactId=my-webapp
```

请注意，这些都必须在一行。这将创建一个目录名**my-webapp**包含以下项目描述：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-webapp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>my-webapp</finalName>
  </build>
</project>
```

笔记==<packaging>==元素- 这告诉Maven去构建一个WAR。改变这个webapp项目的目录和试试：

```
mvn clean package
```

你将看到`target/my-webapp.war`已经构建好了，所有按步骤执行了。

怎么同时建立多个项目？

处理多个模块的概念是建立在Maven。在本节中，我们将展示如何建立上面的WAR，并包括以前的jar，以及在一个步骤。

首先，我们需要在上面其他两项目加上父目录和pom.xml文件，所以它看起来应该像这样：

```
+-- pom.xml
+-- my-app
|   +- pom.xml
|   +- src
|       +- main
|           +- java
+-- my-webapp
|   +- pom.xml
|   +- src
|       +- main
|           +- webapp
```

你创建的POM文件应该包含以下内容：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-app</module>
    <module>my-webapp</module>
```

```
</modules>

</project>
```

我们将需要一个在依赖的JAR来自webapp，所以加入这个到 my-webapp/pom.xml

```
...
<dependencies>
  <dependency>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  ...
</dependencies>
```

最后，添加以下 ==<parent>==元素的 pom.xml文件中其他子目录：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>

  ...
```

现在试着。。。从最顶级目录执行：

```
mvn clean install
```

这个WAR现在已经创建了 my-webapp/target/my-webapp.war，并JAR被包含了：

```
$ jar tvf my-webapp/target/my-webapp-1.0-SNAPSHOT.war
0 Fri Jun 24 10:59:56 EST 2005 META-INF/
222 Fri Jun 24 10:59:54 EST 2005 META-INF/MANIFEST.MF
```

```
0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/
0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/
0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/
3239 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/pom.xml
0 Fri Jun 24 10:59:56 EST 2005 WEB-INF/
215 Fri Jun 24 10:59:56 EST 2005 WEB-INF/web.xml
123 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/pom.properties
52 Fri Jun 24 10:59:56 EST 2005 index.jsp
0 Fri Jun 24 10:59:56 EST 2005 WEB-INF/lib/
2713 Fri Jun 24 10:59:56 EST 2005 WEB-INF/lib/my-app-1.0-SNAPSHOT.jar
```

怎么工作的？首先，父级POM创建（叫做app），有一个**pom**的packaging和一列表的模块定义。这告诉Maven所有执行以上操作设置项目不只是当前这个（从写行为，你可以使用`==non-recursive==`命令选项）

接着，我们告诉WAR它需要**my-app**的JAR。做一些事情：它可以在classpath的WAR获取任何源码(在这种情况下没有)，它确定JAR构建总是在构建WAR之前，和它指示这WAR插件包含JAR在类库目录。

你可能已经注意到，**junit-4.11.jar**是依赖，但并没有结束战争。这样做的原因是`<scope>test</scope>` 它只用于测试，所以在Web应用程序中不包含作为编译时依赖的应用程序。

最后一步是包含父定义。这是扩展元你可能熟悉Maven 1不同：它确保POM可以位于即使项目分别分布于其父通过查找仓库。

不像 Maven 1.0，它不需要你执行 **run**成那些步骤——你可以自己运行包，而反应器中的工件将使用于目标目录而不是本地存储库。

你可以使用你的IDE工作空间的顶级目录创建。。。


```
mvn idea:idea
```

原创文章，转载请注明： 转载自[并发编程网 – ifeve.com](#) **本文链接地址：** [《Maven官方指南》30分钟入门](#)

0

About

Latest Posts



魔术师Carvendy

尘世间一小小Javaer ,喜欢coding和思考。

★[添加本文到我的收藏](#)

Related Posts:

- 1. [《Maven官方指南》可选的依赖和依赖排除](#)
- 2. [《Maven官方指南》Maven使用Ant指南](#)
- 3. [《Maven官方指南》创建一个站点](#)
- 4. [《Maven官方指南》Maven 类加载指南](#)
- 5. [《Maven官方指南》迁移引导](#)
- 6. [《Maven官方指南》使用Maven 2 为不同环境构建](#)
- 7. [《Maven官方文档》创建Archetype](#)
- 8. [《Maven官方指南》jdk1.4项目使用JDK1.5构建指南](#)
- 9. [《Maven官方指南》配置档案插件指南](#)
- 10. [《Maven官方指南》权限和发布设置](#)
- 11. [《Maven官方指南》生成源文件](#)
- 12. [《Maven官方指南》Maven 配置](#)
- 13. [《Maven官方指南》使用扩展](#)
- 14. [《Maven官方指南》创建装配](#)
- 15. [《Maven官方指南》模型指南](#)

Write comment

Comments RSS

Trackback are closed

Comments (0)

No comments yet.

You must be [logged in](#) to post a comment.

[《Spring 5 官方文档》21. 与其他Web框架集成](#)

[《Kafka官方文档》实现](#)

