**简介**

本文将介绍基于 Apache Maven 3 的项目构建的基本概念和方法。Maven 是一套标准的项目构建和管理工具，使用统一规范的脚本进行项目构建，简单易用，摒弃了 Ant 中繁琐的构建元素，并具有较高的可重用性。读完本文，你将了解 Maven 的基本概念和使用它进行项目构建的基本方法。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**什么是 Maven**

Maven 是 Apache 组织下的一个跨平台的项目管理工具，它主要用来帮助实现项目的构建、测试、打包和部署。Maven 提供了标准的软件生命周期模型和构建模型，通过配置就能对项目进行全面的管理。它的跨平台性保证了在不同的操作系统上可以使用相同的命令来完成相应的任务。Maven 将构建的过程抽象成一个个的生命周期过程，在不同的阶段使用不同的已实现插件来完成相应的实际工作，这种设计方法极大的避免了设计和脚本编码的重复，极大的实现了复用。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**Maven vs Ant**

Ant 也是 Apache 组织下的一个跨平台的项目构建工具，它是一个基于任务和依赖的构建系统，是过程式的。开发者需要显示的指定每一个任务，每个任务包含一组由 XML 编码的指令，必须在指令中明确告诉 Ant 源码在哪里，结果字节码存储在哪里，如何将这些字节码打包成 JAR 文件。Ant 没有生命周期，你必须定义任务和任务之间的依赖，还需要手工定义任务的执行序列和逻辑关系。这就无形中造成了大量的代码重复。

Maven 不仅是一个项目构建工具还是一个项目管理工具。它有约定的目录结构（表 1）和生命周期，项目构建的各阶段各任务都由插件实现，开发者只需遵照约定的目录结构创建项目，再配置文件中生命项目的基本元素，Maven 就会按照顺序完成整个构建过程。Maven 的这些特性在一定程度上大大减少了代码的重复。

**表 1. Maven 目录结构**

| **src/main/java** | **Application/Library sources** |
| --- | --- |
| src/main/resources | Application/Library resources |
| src/main/filters | Resource filter files |
| src/main/assembly | Assembly descriptors |
| src/main/config | Configuration files |
| src/main/scripts | Application/Library scripts |
| src/main/webapp | Web application sources |
| src/test/java | Test sources |
| src/test/resources | Test resources |
| src/test/filters | Test resource filter files |
| src/site | Site |
| LICENSE.txt | Project's license |
| README.txt | Project's readme |

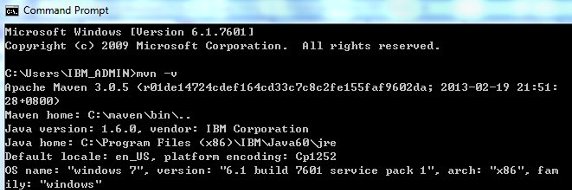
[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**Maven 的安装和基本设置**

本文介绍 Windows 平台上 Maven 的安装。Maven 3 需要运行在 JDK1.4 以上的版本上。

1. 首先确保系统中已经安装 JDK 并正确设置了 Java 环境变量。
2. Maven 的下载地址为：<http://maven.apache.org/download.cgi>。将下载下来的文 件解压到指定的目录中，例如 C:\Maven-3.0。
3. 设置环境变量。打开我的电脑 > 属性 > 高级 > 环境变量，新建系统变量 Maven\_HOME，设置变量值为 Maven 安装目录：C:\Maven-3.0。选择“系统变量”中变量名为“Path”的环境变量，双击该变量，把 Maven 安装路径中 bin 目录的绝对路径，添加到 Path 变量的值中，并使用半角的分号和已有的路径进行分隔：C:\Maven-3.0\bin。
4. 配置完成后，在 Windows DOS 窗口中输入 mvn -v 验证 Maven 安装成功与否，如果安装成功，运行结果会正确显示版本号等安装信息，如图 1。

**图 1. 安装信息**



[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

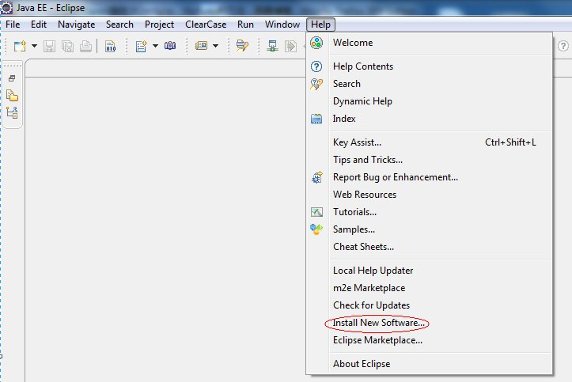
**在 Eclipse3.6 中创建 maven 项目**

* **安装 M2eclipse 插件**

在 Eclipse 中集成 Maven 插件能极大的方便创建 Maven project 并对其进行操作。使用以下的步骤来完成 M2eclipse 插件的安装：

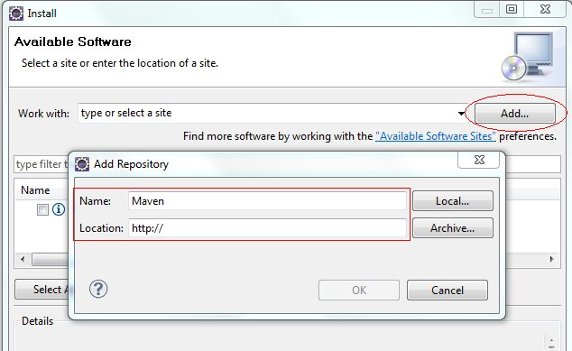
1. 在 Eclipse 菜单栏中选择 Help > Install New Software，如图 2。

**图 2. 安装步骤 1**



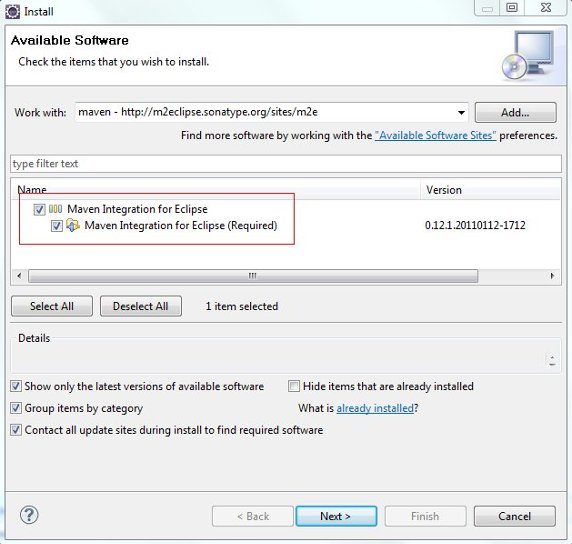
1. 在弹出的对话框中单击 add 按钮，在弹出的 Add Repository 对话框的 Name 和 Location 字段中分别输入 maven 和 http://m2eclipse.sonatype.org/sites/m2e，如图 3，点击 OK。

**图 3. 安装步骤 2**



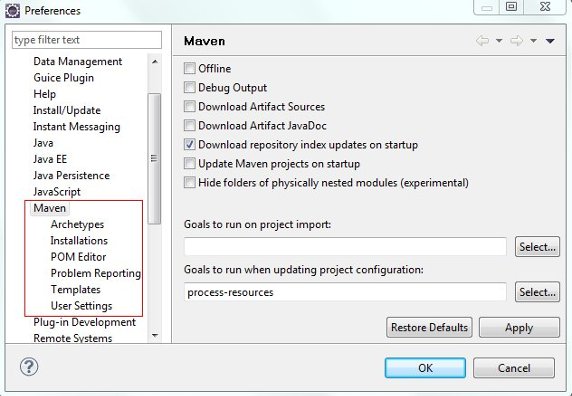
1. 在弹出的资源列表中选择 M2eclipse，点击 next 按照提示进行安装，如图 4。

**图 4. 安装步骤 3**



1. 安装完成后重启 Eclipse，打开 Window > Preferences 检查 maven 插件是否安装成功，如图 5。

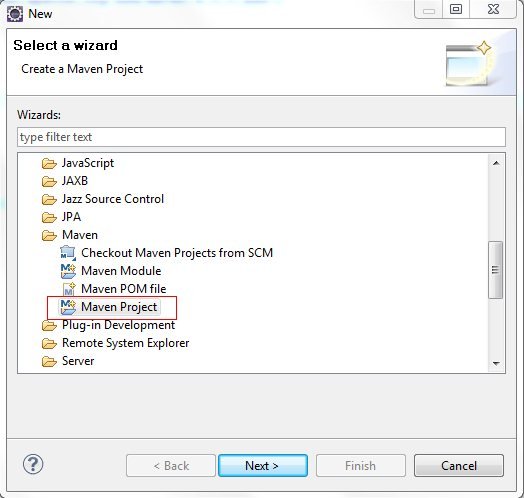
**图 5. 验证 Maven 插件**



* **创建 Maven Project**

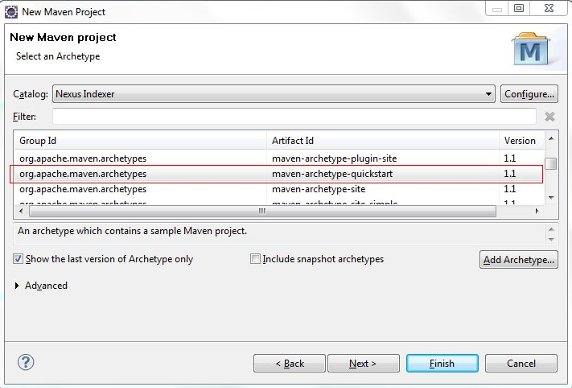
1. 点击 Eclipse 菜单栏 File->New->Ohter->Maven，在弹出的对话框中选择 Maven Project，点击下一步，如图 6。

**图 6. 创建步骤 1**



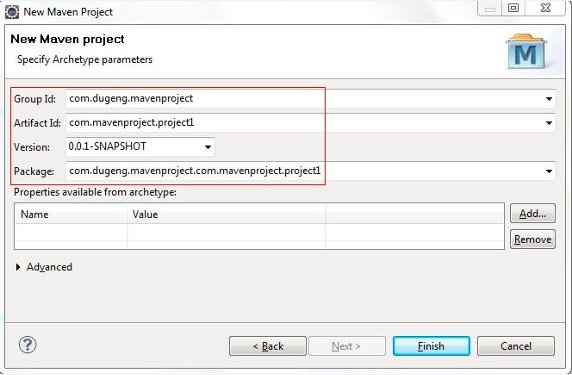
1. 选择 maven-archetype-quickstart，点击下一步，如图 7。

**图 7. 创建步骤 2**



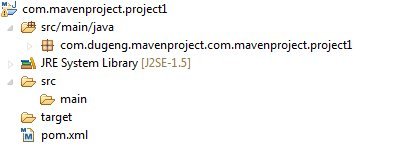
1. 在弹出的对话框中，填写 GroupId, ArtfactId,Version 等信息，他们的具体含义将在后文中介绍，点击 Finish，如图 8。

**图 8. 创建步骤 3**



通过以上步骤我们生成了一个基本的 Maven Project，打开 eclipse 中的 Package Explorer，看到它的结构如图 9 所示。

**图 9. Maven Project 实例**



[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**项目对象模型 POM-Maven 的灵魂**

POM 即 Project Object Module，项目对象模型，在 pom.xml 文件中定义了项目的基本信息、源代码、配置文件、开发者的信息和角色、问题追踪系统、组织信息、项目授权、项目的 url、以及构建项目所用的插件，依赖继承关系。开发人员需按照 maven 定义的规则进行 POM 文件的编写，清单 1 为一个 POM 文件示例。

**清单 1. POM 文件示例**

<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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<! – The Basics – >

<groupId> … </groupId>

<artifactId> … </artifactId>

<version> … </version>

<packaging> … </packaging>

<dependencies> … </dependencies>

<parent> … </parent>

<dependencyManagement> … </dependencyManagement>

<modules> … </modules>

<properties> … </properties>

<! – Build Settings – >

<build> … </build>

<reporting> … </reporting>

<! – More Project Information – >

<name> … </name>

<description> … </description>

<url> … </url>

<inceptionYear> … </inceptionYear>

<licenses> … </licenses>

<organization> … </organization>

<developers> … </developers>

<contributors> … </contributors>

<! – Environment Settings – >

<issueManagement> … </issueManagement>

<ciManagement> … </ciManagement>

<mailingLists> … </mailingLists>

<scm> … </scm>

<prerequisites> … </prerequisites>

<repositories> … </repositories>

<pluginRepositories> … </pluginRepositories>

<distributionManagement> … </distributionManagement>

<profiles> … </profiles>

</project>

在每个 POM 文件中都含有的元素是该 project 的坐标，它包含三个基本元素。

groupId 定义了项目属于哪个组，这有助于在大的范围上区别项目。artifactId 定义了这个项目在组中唯一的 ID。name 是一个用户友好的项目名称。

除了项目坐标外，modelVersion 指定 POM 模型的版本，version 指明当前项目的版本，packaging 指定了项目发布时的打包类型。

在下文中提及的插件，依赖，生命周期等也都有相应的 POM 元素在文件中有所体现。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**Maven 插件和仓库**

Maven 本质上是一个插件框架，它的核心并不执行任何具体的构建任务，仅仅定义了抽象的生命周期，所有这些任务都交给插件来完成的。每个插件都能完成至少一个任务，每个任务即是一个功能，将这些功能应用在构建过程的不同生命周期中。这样既能保证拿来即用，又能保证 maven 本身的繁杂和冗余。

将生命周期的阶段与插件目标相互绑定，就可以在特定的阶段完成具体的构建任务。例如清单 2 中的代码就是要在 validate 这个阶段执行 maven-antrun-plugin 的 run 目标，具体的任务在 <target></target> 元素中定义。

**清单 2. 插件**

<plugins>

<plugin>

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

<artifactId>maven-antrun-plugin</artifactId>

<version>1.6</version>

<executions>

<execution>

<id>version</id>

<phase>**validate**</phase>

<configuration>

<target>

具体任务

</target>

</configuration>

<goals>

<goal> run </goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

Maven 项目中的插件，依赖和项目构建的输出都可以由 Maven 的坐标进行唯一的区分，基于这种机制，Maven 将所有项目的构件文件放置在一个统一的位置，也就是 Maven 仓库。所有 Maven 项目可以从同一个 Maven 仓库中获取自己所需要的依赖 JAR，这节省了磁盘资源。实际的 Maven 项目中不需要存储依赖的文件，只需要在 POM 文件中生成依赖关系，在构建的时候 Maven 就会自动去仓库中下载。

在安装了 Maven 的机器上，会生成一个 ~\.m2\repository 目录，这个目录被称为本地仓库，当 Maven 查找需要的依赖时，首先会在本地查找，如果本地仓库中存在，则直接使用，否则 Maven 回去远程仓库查找，查找到后下载到本地进行使用。远程中央仓库的地址为 http://repo1.maven.org/。当然还有一些镜像仓库可供使用，有兴趣的读者可以参考 [Maven 官方网站的相关介绍](http://maven.apache.org/guides/mini/guide-mirror-settings.html%20%E3%80%82)。

当个人所在的网络无法访问公共的 Maven 仓库时，可以在 settings.xml 中设置代理服务器。打开 ~\.m2\settings.xml，如果没有则复制 $Maven\_HOME/conf/settings.xml 到此路径下，加入清单 3 中的代码：

**清单 3. 代理**

<proxies>

<proxy>

<active>true</active>

<protocol>http</protocol>

<host> 代理地址 </host>

<port>8080</port>

<username> 用户名 </username>

<password> 密码 </password>

</proxy>

</proxies>

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**依赖、聚合和继承**

* **依赖**

我们项目中依赖的 Jar 包可以通过依赖的方式引入，通过在 dependencies 元素下添加 dependency 子元素，可以声明一个或多个依赖。通过控制依赖的范围，可以指定该依赖在什么阶段有效。Maven 的几种依赖范围：

**表 2. 依赖范围**

| **名称** | **有效范围** |
| --- | --- |
| compile | 编译，测试，运行。默认的依赖范围。 |
| test | 测试，如 Junit。 |
| runtime | 运行，如 JDBC。 |
| provided | 编译，测试，如 ServletAPI。 |
| system | 编译，测试，依赖于系统变量。 |

清单 4 中表示引入对 Junit 的依赖 , 这个依赖关系产生作用的阶段是 <scope>test</scope>。

**清单 4. 依赖**

<dependency>

<groupId> </groupId>

<artifactId> </artifactId>

<version> </version>

<optional>true<optional>

</dependency>

依赖是具有传递性的，例如 Project A 依赖于 Project B，B 依赖于 C，那么 B 对 C 的依赖关系也会传递给 A，如果我们不需要这种传递性依赖，也可以用 <optional> 去除这种依赖的传递，如清单 5。

**清单 5. 选择性依赖**

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.1.1</version>

<optional>true<optional>

</dependency>

假设第三方的 jar 包中没有使用 <optional> 来去除某些依赖的传递性，那么可以在当前的 POM 文件中使用 <exclusions> 元素声明排除依赖，exclusions 可以包含一个或者多个 exclusion 子元素，因此可以排除一个或者多个传递性依赖。如清单 6。

**清单 6. 排除依赖**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

* **聚合**

现实中一个项目往往是由多个 project 构成的，在进行构建时，我们当然不想针对多个 project 分别执行多次构建命令，这样极容易产生遗漏也会大大降低效率。Maven 的聚合功能可以通过一个父模块将所有的要构建模块整合起来，将父模块的打包类型声明为 POM，通过 <modules> 将各模块集中到父 POM 中。如清单 7，其中 <module></module> 中间的内容为子模块工程名的相对路径。

**清单 7. 聚合**

<modules>

<module>../com.dugeng.project1</module>

<module>../com.dugeng.project2</module>

</modules>

父类型的模块，不需要有源代码和资源文件，也就是说，没有 src/main/java 和 src/test/java 目录。Maven 会首先解析聚合模块的 POM 文件，分析要构建的模块，并通过各模块的依赖关系计算出模块的执行顺序，根据这个潜在的关系依次构建模块。将各子模块聚合到父模块中后，我们就可以对父模块进行一次构建命令来完成全部模块的构建。

* **继承**

在面向对象的编程中我们学会了继承的概念，继承是可重用行即消除重复编码的行为。Maven 中继承的用意和面向对象编程中是一致的。与聚合的实现类似，我们通过构建父模块将子模块共用的依赖，插件等进行统一声明，在聚合和继承同时使用时，我们可以用同一个父模块来完成这两个功能。

例如将 com.dugeng.parent 这个模块声明为 project1 和 project2 的父模块，那么我们在 project1 和 2 中用如下代码声明父子关系，如清单 8：

**清单 8. 继承**

<parent>

<groupId>com.dugeng.mavenproject</groupId>

<artifactId>com.dugeng.parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

<relativePath>../com.dugeng.parent/pom.xml</relativePath>

</parent>

由于父模块只是用来声明一些可共用的配置和插件信息，所以它也像聚合模块一样只需要包括一个 POM 文件，其它的项目文件如 src/main/java 是不需要的。

聚合和继承存在一些共性和潜在的联系，在实际的应用中，经常将聚合模块的父模块和继承的父模块定义为同一个。

并不是所有的 POM 元素都可以被继承，表 3 是一个可继承的元素列表。

**表 3. 可继承元素列表**

| **名称** | **描述** |
| --- | --- |
| groupId | 项目组 ID |
| version | 项目版本 |
| description | 描述信息 |
| organization | 组织信息 |
| inceptionYear | 创始年份 |
| url | 项目的 url 地址 |
| developers | 开发者 |
| contributors | 贡献者信息 |
| distributionManagerment | 部署信息 |
| issueManagement | 缺陷跟踪系统 |
| ciManagement | 持续继承信息 |
| scm | 版本控制信息 |
| mailingList | 邮件列表信息 |
| properties | 自定义的属性 |
| dependencies | 依赖配置 |
| dependencyManagement | 依赖管理配置 |
| repositories | 仓库配置 |
| build | 源码目录，插件管理等配置 |
| reporting | 报告配置 |

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**Maven 属性**

在 POM 文件中常常需要引用已定义的属性以降低代码的冗余，提高代码的可重用性，这样不仅能降低代码升级的工作量也能提高代码的正确率。有些属性是用户自定义的，有些属性是可以直接引用的已定义变量。

Maven 的可用属性类型可分为 5 种，它们分别是：

1. 内置属性。这种属性跟 Maven Project 自身有关，比如要引入当前 Project 的版本信 息，那么只需要在使用的位置引用 ${version} 就行了。
2. Setting 属性。上文中已经提到 Maven 自身有一个 settings.xml 配置文件，它里面含有包括仓库，代理服务器等一些配置信息，利用 ${settings.somename} 就可以得到文件里相应元素的值。
3. POM 属性。这种属性对应 POM 文件中对应元素的值，例如 ${project.groupId} 对应了 <groupId></groupId> 中的值，${project.artifactId} 对应了 <artifactId> </ artifactId > 中的值。
4. 系统环境变量。可以使用 env.${name} 来获得相应 name 对应的环境变量的值，例如 ${env.JAVA\_HOME} 得到的就是 JAVA\_HOME 的环境变量值。
5. 用户自定义变量。这种类型的变量是使用最频繁和广泛的变量，完全由用户自己定义。在 POM 文件中加入 <properties> 元素并将自定义属性作为其子元素。格式如清单 9。

**清单 9. 自定义属性**

<properties>

<path>../../sourcecode</path>

</properties>

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**Maven 3 的新特性**

Maven 3 在性能和灵活性方面都比 Maven2 有了很大提升，它的新特性总结起来有以下几点：

1. 兼容低版本 Maven，也就是向后兼容，因此用户可以将 Maven2 的项目移植到 Maven3 上来。

2. 性能优化。CPU 利用率更高，内存消耗更小，经过优化的 Maven3 比 Maven2 构建速度快出 50% 以上，这对于构建大型项目的开发者来说无疑会节省大量的时间。

3. 在早先的版本中，开发者必须在子模块中指定父版本，当进行代码的迁移或升级时，这会带来额外的维护工作，Maven3.1 将会消除在子模块上指定父版本的需要。

4.Maven3 改善了错误报告，它会在错误报告中提供指向 Maven Wiki 页面的链接，这样开发者可以方便的查看更全面的错误描述和可能的原因。

5. 增加了 Maven Shell，通常我们可以在系统自带的 console 里执行 Maven 命令，但是通过自安装的 Maven Shell 可以提高生成速度，它是一个是 Maven 的命令行接口工具，可以缓存解析过的 POM，避免了重复调用 Maven 的启动成本。Maven Shell 不属于 Maven 发行包的一部分，需要单独下载。

6. M2Eclipse 实现了 Maven 和 Eclipse 的集成，与一个使用更广泛的 IDE 进行集成从而为开发者带来的便利是不言而喻的。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-lo-maven/#ibm-pcon)

**结束语**

Maven 有着许多实用的特点，它使用了标准的目录结构和部署。这就使得开发人员能够适应不同的项目，并且不用学习任何结构方面新的东西，也不用掌握特殊的指令来构建结构。当然，Maven 的使用还不够普及，相信随着时间的推移，它的功能会更完善，使用的人群也会越来越广泛。