**开始之前**

现代软件开发基于健壮的企业级开源技术，它需要一类新的构建工具和项目协作工具。Apache Maven 2 的核心引擎旨在简化往往十分复杂的大型协作性软件项目的构建和管理。在大型项目团队环境中工作是种挑战，而 Maven 2 以友好为设计目的，即便对那些不熟悉这种挑战的开发人员而言，Maven 2 也足够友好。本教程首先围绕单个的新手开发展开，然后逐渐介绍 Maven 2 中的一些协作性的概念和功能。我们鼓励您在本教程提供的介绍之上，探究本文范围之外的 Maven 2 的一些高级功能。

**关于本教程**

本教程将循序渐进地向您介绍有关 Maven 2 的基础概念和实践练习，具体内容包括：

* Maven 2 概览
* 理解 Maven 2 依赖项管理模型
* Maven 2 存储库和 Maven 2 坐标
* Maven 2 生命周期、阶段、插件和 mojo
* 下载和安装 Maven 2
* 实践 Maven 2 —— 您的第一个 Maven 2 项目
* 定制项目对象模型（POM）
* 参与多个项目
* 实践 Maven 2 —— 参与多个项目构建
* 在 Eclipse 3.2 中安装 Maven 2.x Plug-in
* 在 Eclipse 3.2 中使用 Maven 2.x Plug-in

读完本教程，您会理解并欣赏 Maven 2 背后的设计理念。也将进一步熟悉在使用 Maven 2 构建的项目中工作所需的基本技能。这是一张通往 Apache 和 Codehaus 社区中大多数大型项目的通行证。最为重要的是，您将可以把 Maven 2 应用到日常的项目构建和管理活动中。

**先决条件**

您应当大体上熟悉 Java™ 开发。本教程假设您理解构建工具的价值和基本操作，包括依赖项管理和输出打包。要求您能将 Eclipse 3.2 用作 IDE，从而在 Eclipse 中使用 Maven 2.x 插件。如果您熟悉大型开源项目（如 Apache 软件基金会管理下的项目），那就太好了。理解 Java 5 编码（包括泛型）会很有帮助。如果有各种项目构建技术的经验（如 Ant、autoconf、make 和 nmake）也很好，但不做强制性要求。

**系统需求**

为继续学习和试验本教程中的代码，您需要有效安装 [Sun's JDK 1.5.0\_09](http://java.sun.com/javase/downloads/index.jsp) （或其后的版本）或 [IBM JDK 1.5.0 SR3](http://www.ibm.com/developerworks/java/jdk/?S_TACT=105AGX52&S_CMP=cn-t-j)。

针对教程中关于 Eclipse 的 Maven 2.x 的插件部分，需要有效安装 [Eclipse 3.2.1](http://www.eclipse.org/downloads/) 或其后版本。

本教程的推荐系统配置如下：

* 支持上述 JDK/JRE 的系统，有至少 1GB 的主存
* 有至少 20MB 的磁盘空间来安装软件组件和样例

本教程中的说明基于 Microsoft Windows 操作系统。教程中涵盖的所有工具也可以在 Linux® 和 UNIX® 系统上运行。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**Maven 2 概览**

Maven 是一个顶级的 Apache Software Foundation 开源项目，创建它最初是为了管理 Jakarta Turbine 项目复杂的构建过程。从那以后，不论是开源开发项目还是私有开发项目都选择 Maven 作为项目构建系统。Maven 快速地发展着，如今已是第二版，Maven 已经从针对单个复杂项目的定制构建工具成长为广泛使用的构建管理系统，其丰富的功能可以应用于大多数的软件开发场景。

概括来讲，Maven 2 能够：

* 理解项目通常是如何构建的。
* 利用其内嵌的项目知识简化和便利项目构建。
* 利用其内嵌的项目知识来帮助用户理解复杂的项目结构和构建过程中潜在的变数。
* 设计并实现一个经证实的依赖项管理系统，该系统顺应了当今全球化和相互联系的项目团队的需求。
* 利用其内部知识，针对简单项目提供简单的用户体验。
* 对于高级用户来说相当灵活；针对特别的应用场景，可以覆盖其内嵌模型，也可以通过配置、部署元数据或创建自定义插件对其内嵌模型进行改写。
* 全面扩展现有行为之外的场景细节。
* 捕获新出现的最佳实践和各个用户社区间的经确认的共同性，并将它们纳入到 Maven 的内嵌项目知识中，从而不断地得到改进。

**Maven 2 —— 概念性的概览**

为捕捉项目构建知识，Maven 2 依赖于一套发展中的有关事物如何运转的概念性模型。部分模型被硬编码为 Maven 代码库的一部分，这些模型通过新的 Maven 发布版本不断得到精炼。图 1 解释了 Maven 2 的关键模型：

**图 1. Maven 2 对象和操作模型**

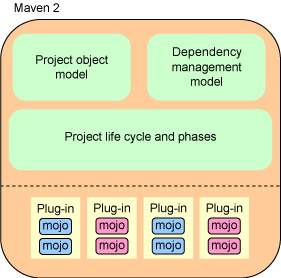


图 1 中的关键组件为：

* **项目对象模型（POM）：** POM 是 Maven 2 中的里程碑式的模型。该模型的一部分已经构建到 Maven 引擎（被亲切地称为*反应堆* ）中，其余部分则通过一个名叫 pom.xml 的基于 XML 的元数据文件来声明。
* **依赖项管理模型：** Maven 对如何管理项目的依赖项很在行。依赖项管理是一片灰色地带，典型的构建-管理工具和系统都未明确涉及。Maven 2 构建了 Maven 依赖项管理模型，该模型能够适应大多数需求。这个模型被证明是有效而高产的模型，目前，主要的开源项目都部署了该模型。
* **构建生命周期和阶段**：和 POM 相关的概念是构建*生命周期* 和*阶段*。这是 Maven 2 的内嵌概念模型和现实物理世界之间的接口。使用 Maven 时，工作主要是通过插件来执行的。在构建周期中，Maven 2 通过遵循一系列定义好的阶段，将这些插件协调起来。

如果您对其中一些概念还有点模糊，请不要担心。接下来的部分会用具体的例子来巩固这些模型背后的概念。

**Maven 2 —— 物理概览**

图 2 揭示了 Maven 2 的操作方式和与之交互的方式，同时显示了它的物理构成。图 2 提供了有关如何与 Maven 进行交互的概览：

**图 2. Maven 2 操作和交互模型**

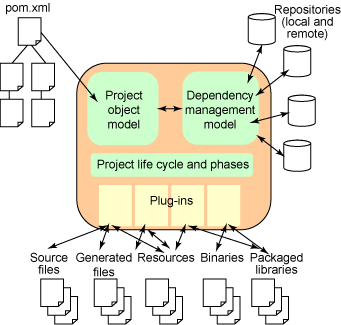


图 2 中，POM 是 Maven 对您的特定项目的理解。这个模型由包含在一系列 pom.xml 文件中的声明性描述构成。这些 pom.xml 文件构成一棵树，每个文件能从其父文件中继承属性。Maven 2 提供一个 Super POM。这个 Super POM 位于层级树的顶端，它包含所有项目的默认通用属性；每个项目的 POM 都从这个 Super POM 处继承。

依赖项被指定为 pom.xml 文件的一部分。Maven 根据其依赖项管理模型解析项目依赖项。Maven 2 在本地存储库和全球存储库寻找依赖性组件（在 Maven 术语里称作*工件* ）。在远程存储库中解析的工件被下载到本地存储库中，以便使接下来的访问可以有效进行。Maven 2 中的这个依赖项解析器可以处理*可递* 依赖项。即，它能有效地解析您的依赖项所依赖的那些依赖项。

Maven 引擎通过*插件* 亲自执行几乎所有的文件处理任务。插件被配置和描述在 pom.xml 文件中。依赖项管理系统将插件当作工件来处理，并根据构建任务的需要来下载插件。每个插件都能和生命周期中的不同阶段联系起来。Maven 引擎有一个状态机，它运行在生命周期的各个阶段，在必要的时候调用插件。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**理解 Maven 2 依赖项管理模型**

在有效利用 Maven 2 之前，您需要理解 Maven 2 依赖项管理模型是如何运行的。

依赖项开发适应于这样的项目，其软件组件（称作*模块* ）是由不同的项目团队开发的。它支持持续独立开发，也支持对所有依赖模块进行精炼。

这个团队协作场景在通过 Internet 建立和维护的开源项目中十分常见，由于内部开发大受开源或外包世界的冲击和影响，这种场景在合作开发的圈子里日益盛行。

**解析项目依赖项**

Maven 2 依赖项管理引擎帮助解析构建过程中的项目依赖项。

**Maven 本地存储库和远程存储库**

Maven 2 本地存储库是磁盘上的一个目录，通常位于*HomeDirectory*/.m2/repository。这个库扮演着高性能本地缓存的角色，存储着在依赖项解析过程中下载的工件。远程存储库要通过网络访问。可以在 settings.xml 配置文件中维护一个远程存储库列表以备使用。

实践中，依赖项在 pom.xml 文件内的 <dependencies> 元素中指定，并作为 POM 的一部分注入到 Maven 中。

项目依赖项存储在*存储库服务器*（在 Maven 术语中简单地称之为*存储库* ）上。要成功的解析依赖项，需要从包含该工件的存储库里找到所需的依赖性工件。

**通过 settings.xml 配置 Maven**

可以在一个 settings.xml 文件中指定影响 Maven 操作的配置属性。默认的设置文件是*MavenInstallationDirectory*/conf/settings.xml。Maven 2 用户可通过维护 *UserHomeDirectory*/.m2/settings.xml 来覆盖一些配置属性。参见 [Maven 设置参考](http://maven.apache.org/settings.html)，获取更多有关可配置设置的信息。

基于 POM 中的项目依赖项信息，该依赖项解析器试图以下列方式解析依赖项：

1. 检查本地存储库中的依赖项。
2. 检查远程存储库列表中的依赖项。
3. 如果前两步失败，则报告一个错误。

默认情况下，第二步中所涉及的第一个远程存储库是一个能在全球访问的集中式 Maven 2 存储库，它包含了最流行的开源项目的一些工件。在内部开发中，可以设置额外的远程存储库来包含从内部开发模块中发布的工件。可以使用 settings.xml 中的 <repositories> 元素来配置这些额外的远程存储库。

**确保单个的工件**

将 Maven 2 用于项目构建时，依赖项解析通过一个集中的存储库确保只存在一个依赖性工件，而不考虑有多少项目或子项目引用该工件。这是多模块项目构建中一个重要的属性，因为包含多个工件会导致一些项目一致性和集成方面的问题。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**存储库和坐标**

Maven 2 存储库存储 Maven 在一个项目的依赖项解析中使用过的工件集。在本地磁盘上访问本地存储库，通过网络访问远程存储库。

工件通常被打包成包含二进制库或可执行库的 JAR 文件。这被认为是工件的一个*类型*。但在实践中，工件也可以是 WAR、EAR 或其他代码捆绑类型。

Maven 2 利用操作系统的目录结构对存储在存储库中的工件集进行快速索引。这个存储库索引系统依赖于这种能力来通过工件的*坐标* 惟一标识工件。

**Maven 坐标**

Maven 坐标是一组可以惟一标识工件的三元组值。坐标包含了下列三条信息：

* **组 ID**：代表制造该工件的实体或组织。例如，com.ibm.devworks 就是一个组 ID。
* **工件 ID**：实际的工件的名称。例如，主体类名为 OpsImp 的项目也许会用 OpsImp 作为其工件 ID。
* **版本**：该工件的版本号。支持的格式为 *mmm.nnn.bbb-qqqqqqq-dd*，其中， *mmm* 是主版本号，*nnn* 是次版本号，*bbb* 代表其 bug 修复水平。*qqqqq*（限定词）或 *dd*（构建号）也能添加到版本号中，这两项是可选项。

对 Maven 坐标的使用贯穿于 Maven 配置文件和 POM 文件中。例如，要在命名为 OpsImp 的模块上指定项目依赖项（在 1.0-SNAPSHOT 级别），pom.xml 文件应包含清单 1 所示的部分：

**清单 1. OpsImp 样例模块的 Maven 坐标**

<dependencies>

<dependency>

**<groupId>com.ibm.devworks</groupId>**

**<artifactId>OpsImp</artifactId>**

**<version>1.0-SNAPSHOT</version>**

</dependency>

</dependencies>

特别限定词 SNAPSHOT 告诉 Maven 2：该项目或模块还处于开发状态中，它应该获取最新版的可用工件。

要将该项目指定为依赖于 JUnit 进行单元测试，可以将 JUnit 3.8.1 的坐标作为一个依赖项添加到该项目的 pom.xml 文件中，如清单 2 所示：

**清单 2. JUnit 依赖项的 Maven 坐标**

<dependencies>

<dependency>

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>3.8.1</version>**

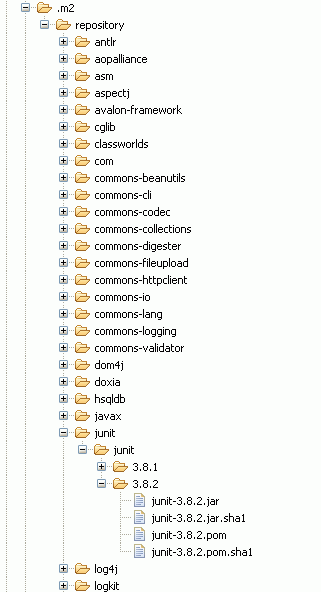
</dependency>

</dependencies>

**深入 Maven 存储库**

由于 Maven 存储库是普通的目录树，所以可以很容易地看到工件是如何存储到磁盘上的。图 3 是本地存储库的一部分，显示了 JUnit 3.8.1 工件的位置：

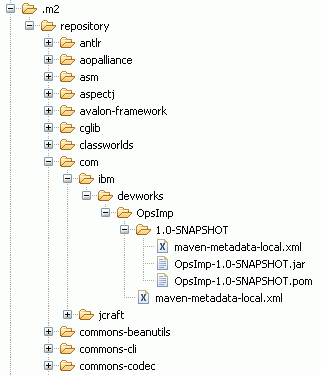
**图 3. Maven 2 存储库内幕**



从图 3 中可以看出，Maven 维护了一个工件的 POM 文件，同时也为该工件和其存储库中的 POM 维护了检验和散列。当工件在存储库间转移时，这些文件帮助确保工件的完整性。该工件已由 Maven 的依赖项管理引擎从中央存储库下载并放置到本地存储库中。

在图 4 中，坐标为 com.ibm.devworks/OpsImp/1.0-SNAPSHOT 的工件显示在本地存储库中。该工件和 POM 文件一起存放在存储库。在本例中，该工件在本地安装。

**图 4. 本地存储库中的 OpsImp 工件**



[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**Maven 2 生命周期、阶段、插件和 mojo**

Maven 通过插件动作完成大多数构建任务。可以把 Maven 引擎认为是插件动作的协调器。

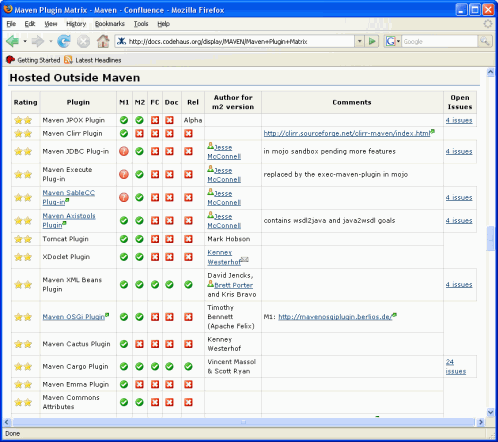
**插件中的 Mojo**

插件是适应 Maven 的插件框架的软件模块。现在，可以使用 Java、Ant 或 Beanshell 来创建自定义插件。插件中的每个任务称作一个 *mojo*。有时，插件也被视为一套相关的 mojo。创建自定义的 Maven 2 插件超出了本教程的范围；参见 [参考资料](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#resources)，以获取更多信息。

Maven 2 是预先打包好的，以便于下载，它和许多常用插件一起使用。大多数典型开发任务不需要使用额外插件。

在开始编写自己的插件前，您应该先参考一下列出流行 Maven 2 插件的 Web 站点（参见 [参考资料](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#resources)），看一下您需要的插件是不是已经有了。图 5 显示了 Maven Plugin Matrix（参见 [参考资料](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#resources)），它提供了许多可用插件的兼容性信息：

**图 5. Maven Plugin Matrix**



**将 mojo 绑定到生命周期各阶段**

Maven 引擎在执行构建生命周期中相应的阶段时，执行插件中的 mojo（构建任务）。插件的 mojo 和生命周期中的阶段间的关联叫做*绑定* 。插件开发人员能够灵活地将一个或多个生命周期阶段和一个插件关联起来。

**默认的生命周期的各阶段**

Maven 对构建生命周期的固定理解包含了许多不同的阶段。表 1 简短地描述了各个阶段：

**表 1. Maven 2 默认生命周期的各阶段**

| **生命周期阶段** | **描述** |
| --- | --- |
| 验证 | 确保当前配置和 POM 的内容是有效的。这包含对 pom.xml 文件树的验证。 |
| 初始化 | 在执行构建生命周期的主任务之前可以进行初始化。 |
| 生成源码 | 代码生成器可以开始生成在以后阶段中处理或编译的源代码。 |
| 处理源码 | 提供解析、修改和转换源码。常规源码和生成的源码都可以在这里处理。 |
| 生成资源 | 可以生成非源码资源。通常包括元数据文件和配置文件。 |
| 处理资源 | 处理非源码资源。修改、转换和重定位资源都能在这阶段发生。 |
| 编译 | 编译源码。编译过的类被放到目标目录树中。 |
| 处理类 | 处理类文件转换和增强步骤。字节码交织器和常用工具常在这一阶段操作。 |
| 生成测试源码 | mojo 可以生成要操作的单元测试代码。 |
| 处理测试源码 | 在编译前对测试源码执行任何必要的处理。在这一阶段，可以修改、转换或复制源代码。 |
| 生成测试资源 | 允许生成与测试相关的（非源码）资源。 |
| 处理测试资源 | 可以处理、转换和重新定位与测试相关的资源。 |
| 测试编译 | 编译单元测试的源码。 |
| 测试 | 运行编译过的单元测试并累计结果。 |
| 打包 | 将可执行的二进制文件打包到一个分布式归档文件中，如 JAR 或 WAR。 |
| 前集成测试 | 准备集成测试。这种情况下的集成测试是指在一个受到一定控制的模拟的真实部署环境中测试代码。这一步能将归档文件部署到一个服务器上执行。 |
| 集成测试 | 执行真正的集成测试。 |
| 后集成测试 | 解除集成测试准备。这一步涉及测试环境重置或重新初始化。 |
| 检验 | 检验可部署归档的有效性和完整性。过了这个阶段，将安装该归档。 |
| 安装 | 将该归档添加到本地 Maven 目录。这一步让其他可能依赖该归档的模块可以使用它。 |
| 部署 | 将该归档添加到远程 Maven 目录。这一步让这个工件能为更多的人所用。 |

Maven 从开源社区中汲取了十多年的项目构建管理经验。很难找到一个构建周期不符合表 1 中的生命周期阶段的软件项目。

启动 Maven 2 引擎后，它会按顺序经历表 1 中的各阶段，执行可能与该阶段绑定的 mojo。每个 mojo 则可以使用 Maven 2 丰富的 POM 支持、依赖项管理，也可以访问执行这一专门任务时的构建状态信息。

调用 Maven 2 引擎时，可以将一个生命周期阶段指定为命令行参数。该引擎一直执行到指定的阶段（包括该指定的阶段）。包含的阶段中所有的 mojo 都会被触发。

简短地说，这就是 Maven 2 的操作原理。在下一部分里，您将直接面对操作。对 Maven 操作、它的依赖项管理模型和它的 POM 有了基本的理解，您会发现实践 Maven 2 是一项十分简单的练习。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**下载和安装 Maven 2**

根据下列步骤下载和安装 Maven 2：

1. 从 Maven 项目的官方站点下载 Maven 2 二进制文件（参见 [参考资料](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#resources)）。
2. 将二进制发布文件解压到选定的目录。
3. 将 *InstallationDirectory*\bin 目录添加到 PATH 环境变量。

为检验安装，键入 mvn -help 命令。您将看到清单 3 中的帮助页面。

**清单 3. 使用 mvn -help 命令检验安装**

C:\>mvn -help

usage: mvn [options] [<goal(s)>] [<phase(s)>]

Options:

-C,--strict-checksums Fail the build if checksums don't match

-c,--lax-checksums Warn if checksums don't match

-P,--activate-profiles Comma-delimited list of profiles to

activate

-ff,--fail-fast Stop at first failure in reactorized builds

-fae,--fail-at-end Only fail the build afterwards; allow all

non-impacted builds to continue

-B,--batch-mode Run in non-interactive (batch) mode

-fn,--fail-never NEVER fail the build, regardless of project

result

-up,--update-plugins Synonym for cpu

-N,--non-recursive Do not recurse into sub-projects

-npr,--no-plugin-registry Don't use ~/.m2/plugin-registry.xml for

plugin versions

-U,--update-snapshots Update all snapshots regardless of

repository policies

-cpu,--check-plugin-updates Force upToDate check for any relevant

registered plugins

-npu,--no-plugin-updates Suppress upToDate check for any relevant

registered plugins

-D,--define Define a system property

-X,--debug Produce execution debug output

-e,--errors Produce execution error messages

-f,--file Force the use of an alternate POM file.

-h,--help Display help information

-o,--offline Work offline

-r,--reactor Execute goals for project found in the

reactor

-s,--settings Alternate path for the user settings file

-v,--version Display version information

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**实践 Maven 2：您的第一个 Maven 2 项目**

在第一个实践的例子里，您将看到如何用最小的努力使用 Maven 2 构建简单项目。Maven 2 内置的关于 Java 项目的知识消除了其他构建工具也许必须要经历的冗长的配置过程。

**处理数值操作的类**

该例使用了一个处理数值操作的类。这个主体类的源代码（叫做 NumOps）如清单 4 所示。

**清单 4.NumOps 类**

package com.ibm.devworks;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class NumOps {

private List <Operation> ops = new ArrayList

<Operation>();

public NumOps() {

ops.add( new AddOps());

}

public Operation getOp(int i)

{ Operation retval;

if (i > ops.size())

{ retval = null;

} else {

retval = (Operation) ops.get(i);

} return retval;

}

public int size() {

return ops.size();

}

public static void main( String[] args ) {

NumOps nop = new NumOps();

for (int i=0; i < nop.size(); i++) {

System.out.println( "2 " +

nop.getOp(i).getDesc() +

" 1 is " +

nop.getOp(i).op(2,1) );

}

}

}

NumOps 类管理一个能够在两个整数上执行数值操作的对象集。main 方法创建一个 NumOps 实例，然后调用由 NumOps 管理的每个对象，同时分别调用它的 getDesc() 方法和 op() 方法。由 NumOps 管理的所有对象都实现了在 Operation.java 中定义的 Operation 接口，接口代码如清单 5 所示：

**清单 5. Operation 接口**

package com.ibm.devworks;

public interface Operation {

int op(int a, int b);

String getDesc();

}

在这个初始的例子里定义的惟一操作是一个 AddOps 类，如清单 6 所示：

**清单 6. AddOps 类**

package com.ibm.devworks;

public class AddOps implements Operation {

public int op(int a, int b) {

return a+b;

}

public String getDesc() {

return "plus";

}

}

执行 NumOps 类时，会打印出下列输出：

2 plus 1 is 3

**使用 Archetype 来创建初始项目**

要创建能用 Maven 构建的简单 Java 项目所需的一切，可以使用 Archetype 插件，它是 Maven 2 中的标准插件。不同于构建阶段插件，Archetype 在 Maven 项目构建生命周期之外运行，用于*创建* Maven 项目。在您想要包含 NumOps 项目的目录中执行下列命令（将所有命令敲在一行）：

mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes

-DgroupId=com.ibm.devworks -DartifactId=NumOps

该命令为 Archetype 插件提供您模块的坐标：com.ibm.devworks/NumOps/1.0-SNAPSHOT。在此情况下，不需要指定版本，因为 Archetype 插件常默认为 1.0-SNAPSHOT。此命令为项目创建了一个起始的 pom.xml 文件，也创建了规范的 Maven 2 目录结构。您将在本教程源代码下载处的 example1 目录下找到该代码（参见 [下载](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#download)）。

该输出应该与清单 7 类似：

**清单 7. 使用 Maven Archetype 创建 NumOps 项目**

[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'archetype'.

[INFO] -------------------------------------------------------------------------

---

[INFO] Building Maven Default Project

[INFO] task-segment: [archetype:create] (aggregator-style)

[INFO] -------------------------------------------------------------------------

---

[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus

...

[INFO] [archetype:create]

[INFO] Defaulting package to group ID: com.ibm.devworks

[INFO] -------------------------------------------------------------------------

---

[INFO] Using following parameters for creating Archetype: maven-archetype-quicks

tart:RELEASE

[INFO] -------------------------------------------------------------------------

---

[INFO] Parameter: groupId, Value: com.ibm.devworks

[INFO] Parameter: packageName, Value: com.ibm.devworks

[INFO] Parameter: basedir, Value: C:\temp\maven

[INFO] Parameter: package, Value: com.ibm.devworks

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] Parameter: artifactId, Value: NumOps

[INFO] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of debug info from resources from generated POM

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[INFO] Archetype created in dir: C:\temp\maven\NumOps

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 1 second

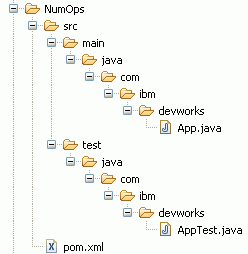
[INFO] Finished at: Sat Dec 02 22:04:02 EST 2006

[INFO] Final Memory: 4M/8M

[INFO] ------------------------------------------------------------------------

Archetype 插件创建了一个目录树、一个 pom.xml 文件和一个占位符 App.java 应用程序文件。它也为单元测试源码创建了一棵目录树，还创建了一个占位符 AppTest.java 单元测试文件。这个项目已经准备就绪。图 6 显示了该 Archetype 插件创建的目录和文件。

**图 6. Archetype 生成的目录和文件**



您只需将 NumOps.java、Operation.java 和 AddOps.java 文件移到 App.java 的位置，并移除 App.java。在下一部分中，将做出一些改变来定制生成的 pom.xm。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**定制 POM**

Maven 2 通过该 pom.xml 文件了解您的项目。该文件由 Archetype 按照 NumOps 生成，如清单 8 所示：

**清单 8. Archetype 生成的 POM - 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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.ibm.devworks</groupId>

<artifactId>NumOps</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>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

请注意：在测试阶段中（通过 <scope> 标记），Archetype 如何定义模块的坐标、如何将类型定义为 JAR 归档以及如何将 JUnit 指定为一个依赖项。要为您的新项目定制这个 pom.xml 文件，请参照清单 9 中突出显示的部分，稍作改动。

**清单 9. 为 NumOps 项目定制生成的 pom.xml 文件**

[点击查看代码清单](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#N10388)

额外的 <build> 标记是必要的，用来覆盖源代码，以达到 Java 代码的水平。默认情况下，采用的是 JDK 1.4，但您的代码使用了泛型，因而需要 JDK 5.0 编译。

**编译定制的项目**

现在可以使用 mvn compile 命令编译 NumOps 项目。 这个命令使 Maven 2 引擎从构建生命周期一直运行到编译阶段，并执行相应的 mojo。您应该看到构建成功的报告，报告中在目标目录树里创建了三个类文件（如清单 10 所示）。如果这是第一次运行，那会花点时间，因为一些依赖项需要经过 Internet 从中央存储库下载。

**清单 10. NumOps 项目中 mvn 编译的输出**

[INFO] Scanning for projects...

[INFO] -------------------------------------------------------------------------

---

[INFO] Building Intro to Maven 2 Example 1

[INFO] task-segment: [compile]

[INFO] -------------------------------------------------------------------------

---

[INFO] [resources:resources]

[INFO] Using default encoding to copy filtered resources.

[INFO] [compiler:compile]

Compiling 3 source files to C:\temp\maven\NumOps\target\classes

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 1 second

[INFO] Finished at: Sat Dec 02 22:52:16 EST 2006

[INFO] Final Memory: 3M/7M

[INFO] ------------------------------------------------------------------------

**添加单元测试**

项目开发的最佳实践通常要求对所有代码模块进行单元测试。Maven 2 为您创建了一个占位符 AppTest.java 单元测试文件。现在，将文件名改为 NumOpsTest.java，请参照清单 11 中突出显示的部分，对生成的单元测试做出改动。也可以从源代码下载处复制单元测试的源代码（参见[下载](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#download)）。

**清单 11. 为项目添加 NumOpsTest 单元测试**

[点击查看代码清单](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#N103CE)

现在可以使用 mvn test 命令运行所有的 mojo 一直到测试阶段。

Maven 2 编译源码和单元测试。然后运行测试，同时报告成功、失败和错误的数目，如清单 12 所示：

**清单 12. 执行 mvn 测试来编译项目和运行单元测试**

[INFO] Scanning for projects...

[INFO] -------------------------------------------------------------------------

---

[INFO] Building Intro to Maven 2 Example 1

[INFO] task-segment: [test]

[INFO] -------------------------------------------------------------------------

---

[INFO] [resources:resources]

[INFO] Using default encoding to copy filtered resources.

[INFO] [compiler:compile]

[INFO] Nothing to compile - all classes are up to date

[INFO] [resources:testResources]

[INFO] Using default encoding to copy filtered resources.

[INFO] [compiler:testCompile]

Compiling 1 source file to C:\temp\maven\NumOps\target\test-classes

[INFO] [surefire:test]

[INFO] Surefire report directory: C:\temp\maven\NumOps\target\surefire-reports

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running com.ibm.devworks.NumOpsTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 2 seconds

[INFO] Finished at: Sat Dec 02 23:04:27 EST 2006

[INFO] Final Memory: 3M/6M

[INFO] ------------------------------------------------------------------------

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**实践 Maven 2：参与多个项目构建**

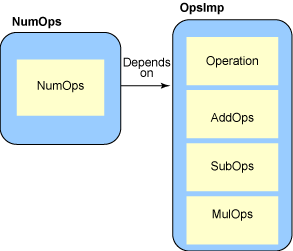
使用 Maven 2 构建和测试简单项目是很直接的。本部分用第二个例子来展示更现实也更为通用的多模块项目。

**扩展 NumOps 样例**

NumOps 的例子将在此处的第二个例子里得到扩展。添加一个新 SubOps 类来支持减法，添加一个新 MulOps 类来支持乘法。

但 Operation 接口和 AddOps 类现在己从 NumOps 项目中移走。相反，它们和新的 SubOps 和 MulOps 类一起放到了一个叫做 OpsImp 的新项目中。图 7 显示了 NumOps 和 OpsImp 项目间的这种关系：

**图 7. NumOps 和 OpsImp 间的关系**



在大型软件开发项目中，子项目和子模块之间常存在依赖性。您可以将这里的这项技术应用到有着多个相互依赖项的任何多模块 Maven 项目中。

清单 13 里的 SubOps 在编码上和 AddOps 类似。这里没有显示的 MulOps 也类似；您可以看一下随附的代码来了解详情（参见 [下载](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#download)）。

**清单 13. 实现 Operation 接口的新 SubOps 类**

package com.ibm.devworks;

public class SubOps implements Operation {

public int op(int a, int b) {

return a-b;

}

public String getDesc() {

return "minus";

}

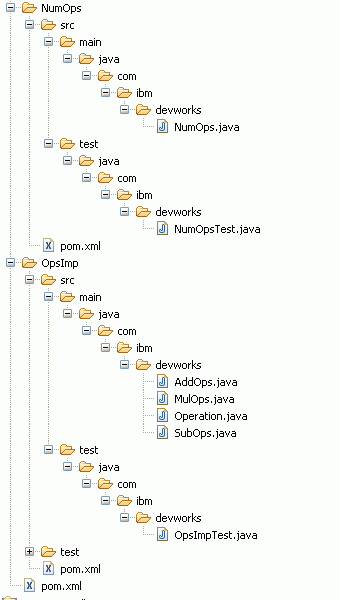
}

现在修改了 NumOps 的构造函数来创建一个 SubOps 实例和一个 MulOps 实例。参见随附的源代码获取详情。

**创建主项目**

为和这两个项目一起运行，主项目创建在比 NumOps 和 OpsImp 的项目目录高一级的目录。NumOps 和 OpsImp 项目都使用标准 Maven 项目目录布局。在最顶部，项目目录只包含一个 pom.xml 文件。图 8 显示了新的子目录结构，紧跟在主目录之下：

**图 8. 多模块项目的目录结构**



可以在分发代码的 example2 子目录中找到这个多模块项目的代码（参见 [下载](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#download)）。清单 14 显示了顶层的 pom.xml 文件：

**清单 14. 多模块项目顶层的 pom.xml 文件**

[点击查看代码清单](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#N1047A)

新代码以粗体突出显示。首先，这个主项目的工件 ID 是 mavenex2，其打包类型是 pom。这向 Maven 2 传递出这样的信号：这是一个多模块项目。

随后，<modules> 标记指定组成此项目的两个模块：NumOps 和 OpsImp。

这个主项目的子模块能从这个 pom.xml 文件中继承属性。说得更具体一点，这些子模块都不需要将 JUnit 声明为一个依赖项，即使它们都包含单元测试。这是因为它们继承了顶层定义的 JUnit 依赖项。

<dependencyManagement> 标记不指定此模块依赖的依赖项。相反，它主要由子模块使用。子模块能指定 <dependencyManagement> 标记中任何条目的依赖项，而无需指定具体的版本号。当项目树更改依赖项的版本号时，这很有用，可以使需编辑的条目数目最小化。在本例中，OpsImp 项目的版本号是使用 ${project.version} 指定的。在执行 Maven 时，这是一个会被相应值所填充的参数。

**从主 POM 中继承**

前进到下一层的 OpsImp 目录，可以找到该模块的 pom.xml 文件，如清单 15 所示：

**清单 15. 新 OpsImp 项目的 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/maven-v4\_0\_0.xsd">

**<parent>**

**<groupId>com.ibm.devworks</groupId>**

**<artifactId>mavenex2</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**</parent>**

<modelVersion>4.0.0</modelVersion>

<artifactId>OpsImp</artifactId>

<packaging>jar</packaging>

</project>

<parent> 元素指定此模块所继承自的主 POM。从父模块中继承极大地简化了这个 pom.xml。只需要覆盖该工件 ID 并打包。此模块继承了其父模块的依赖项：JUnit 模块。

NumOps pom.xml 也从其父模块中继承，也相当简单。这个 pom.xml 显示在清单 16 中：

**清单 16. NumOps 项目中显示 POM 继承关系的 pom.xml**

[点击查看代码清单](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#N104D5)

**发现有效的 POM**

只要从一个高层 POM 中继承，您总能在了解了所有继承元素后发现相对应的 pom.xml 是什么样。显示 “effective POM” 的命令是 mvn help:effective-pom。您将看到一些您还没有指定的元素。这些都是从 Super POM 中继承的。每个项目 pom.xml 都从 Maven 内嵌的 Super POM 中隐式地继承。

NumOps POM 中有意思的一处是将 OpsImp 项目指定为一个依赖项。请注意，在这个依赖项中没有指定任何版本号。在它的父项目的 <dependencyManagement> 元素里已经指定了首选的版本号。

在项目顶层，现在可以用 mvn compile 命令编译全部两个模块，或用 mvn test 命令为两个模块运行单元测试。也可以运行 mvn install 将打包的模块安装到本地目录中。这使得任何依赖于它的模块可以不访问源代码就能解析依赖项。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

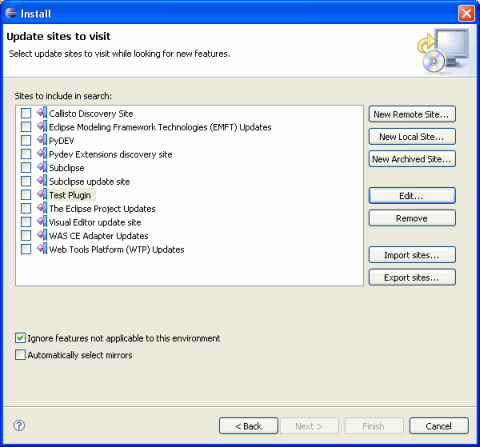
**在 Eclipse 3.2 中安装 Maven 2.x Plug-in**

如果您在日常开发中使用 Eclipse IDE，那么您应该在 Eclipse 中下载和安装 Maven 2.x Plug-in。这个插件有助于在 Eclipse IDE 中操作 Maven 项目。参见 [参考资料](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#resources) 获取有关此插件的项目信息。

可以使用 Eclipse 的软件更新向导在 Eclipse 下安装 Maven 2.X Plug-in：

1. 从 Eclipse 的 **Help** 菜单中选择 **Software Updates>Find and Install...**。
2. 选择 **Search for new features to install**，单击 **Next**。您将看到更新站点向导，如图 9 所示：

**图 9. Eclipse 更新站点向导**



1. 单击 **New Remote Site** 按钮。
2. 在弹出的对话框中，输入 Maven 2.x Plug-in for Eclipse 更新站点，如图 10 所示：

**图 10. Eclipse 新的远程站点**



1. 选择要安装的插件，让安装向导重启工作空间。重启后，您就可以在 Eclipse 中使用 Maven Plug-in 的功能了。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**在 Eclipse 3.2 中使用 Maven 2 Plug-in**

本节涵盖了 Eclipse 中 Maven 2.x Plug-in 的一些常用功能。

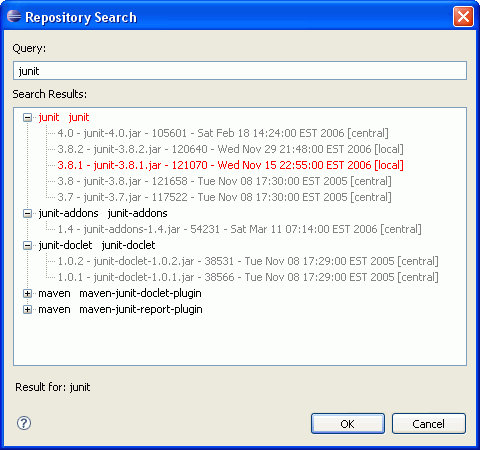
在您的项目可以使用 Maven 2 插件功能前，需要在一个 Eclipse 项目中启用 Maven 属性。右键单击您想要向其中添加 Maven 支持的项目，选择**Maven2>Enable**。

为确保项目的目录结构符合 Maven 的预期，您应先创建 Maven 目录结构（或者手动或者使用 Archetype），然后将该项目添加到 Eclipse 中。

**在存储库中实时搜索依赖项**

使用插件将依赖项添加到 pom.xml 很简单。右键单击该项目的 pom.xml，然后选择 **Maven2>Add Dependency**。这会启动 Repository Search 向导。键入您要寻找的依赖项名称开头的一些字母，该向导会在中央存储库中搜索任何相匹配的工件。相匹配的工件的所有详情都会呈现给您，以帮助您选择依赖项。图 11 显示了搜索 JUnit 工件的结果：

**图 11. Maven Repository Search 向导**

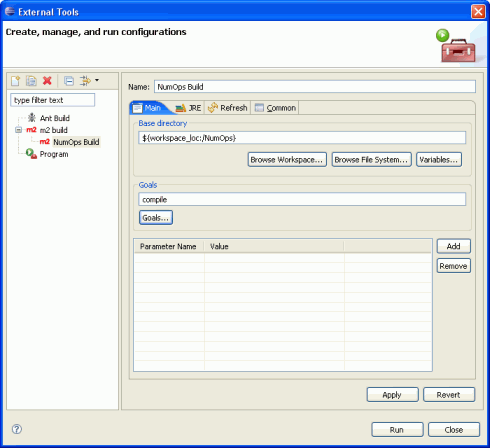


一旦选定了想要的工件的版本，单击 **OK**，插件就会自动地将一个新的 <dependency> 元素添加到 pom.xml 中。

**调用 Maven 构建**

一个构建涉及到能从 Eclipse 中启动的生命周期的任何阶段。首先，确保启用了 Maven 的项目当前是打开的。然后，从 Eclipse 菜单中选择**Run>External Tools>External Tools...**。这将显示 External Tools 向导，如图 12 所示：

**图 12.使用 Eclipse External Tool 向导通过 Maven 构建**



给这个配置文件起名，然后通过单击 **Goals...** 按钮选择一个生命周期阶段。单击 **Run** 运行 Maven。

Maven 输出显示在 Eclipse 的 **Console** 选项卡中。

[**回页首**](http://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html#ibm-pcon)

**结束语**

在本教程中，您：

* 探究了 Maven 2 设计背后的模型和动机
* 理解了所有重要的 Maven POM
* 观察了 Maven 存储库和坐标系统是如何简化复杂依赖项管理的
* 在快捷简单的项目中使用了 Maven 2
* 了解了 Maven 2 如何有助于大型多模块项目
* 实际体验了 Maven 2 Eclipse Plug-in
* 直接观察了 Maven 2 如何便利您的日常项目构建和源码/二进制管理活动

随着软件开发协作的发展，Maven 也会不断发展和改变来适应这种需求。作为大多数大型开源项目的骨干构建工具，它一定能从开发者社区不断提供的改进意见中获益。

一旦理解了 Maven 2 的动机和它想要战胜的挑战，了解 Maven 2 就不困难了。作为构建工具，Maven 2 高效可用，即使是一个十足的新手在一个与世隔绝的地方开发项目，他也会深感 Maven 2 的这一特性。开始在您自己的开发项目中使用 Maven 2 并参加一个开源开发社区吧，很快，您也将能够对 Maven 2 的发展方向和进程有所影响。