第 1 章

Maven 简介

本章内容

- ❑ 何为 Maven
- ❑ 为什么需要 Maven
- ❑ Maven 与极限编程
- ❑ 被误解的 Maven
- ❑ 小结

1.1 何为 Maven

Maven 这个词可以翻译为“知识的积累”，也可以翻译为“专家”或“内行”。本书将介绍 Maven 这一跨平台的项目管理工具。作为 Apache 组织中的一个颇为成功的开源项目，Maven 主要服务于基于 Java 平台的项目构建、依赖管理和项目信息管理。无论是小型的开源类库项目，还是大型的企业级应用；无论是传统的瀑布式开发，还是流行的敏捷模式，Maven 都能大显身手。

1.1.1 何为构建

不管你是否意识到，构建（build）是每一位程序员每天都在做的工作。早上来到公司，我们做的第一件事情就是从源码库签出最新的源码，然后进行单元测试，如果发现失败的测试，会找相关的同事一起调试，修复错误代码。接着回到自己的工作上来，编写自己的单元测试及产品代码，我们会感激 IDE 随时报出的编译错误提示。

忙到午饭时间，代码编写得差不多了，测试也通过了，开心地享用午餐，然后休息。下午先在昏昏沉沉中开了个例会，会议结束后喝杯咖啡继续工作。刚才在会上经理要求看测试报告，于是找了相关工具集成进 IDE，生成了像模像样的测试覆盖率报告，接着发了一封电子邮件给经理，松了口气。谁料 QA 小组又发过来了几个 bug，没办法，先本地重现再说，于是熟练地用 IDE 生成了一个 WAR 包，部署到 Web 容器下，启动容器。看到熟悉的界面了，遵循 bug 报告，一步步重现了 bug……快下班的时候，bug 修好了，提交代码，通知 QA 小组，在愉快中结束了一天的工作。

仔细总结一下，我们会发现，除了编写源代码，我们每天有相当一部分时间花在了编译、运行单元测试、生成文档、打包和部署等烦琐且不起眼的工作上，这就是构建。如果我们现在还手工这样做，那成本也太高了，于是有人用软件的方法让这一系列工作完全自动化，使得软件的构建可以像全自动流水线一样，只需要一条简单的命令，所有烦琐的步骤都能够自动完成，很快就能得到最终结果。

1.1.2 Maven 是优秀的构建工具

前面介绍了 Maven 的用途之一是服务于构建，它是一个异常强大的构建工具，能够帮我们自动化构建过程，从清理、编译、测试到生成报告，再到打包和部署。我们不需要也不应该一遍又一遍地输入命令，一次又一次地点击鼠标，我们要做的是使用 Maven 配置好项目，然后输入简单的命令（如 mvn clean install），Maven 会帮我们处理那些烦琐的任务。

Maven 是跨平台的，这意味着无论是在 Windows 上，还是在 Linux 或者 Mac 上，都可以使用同样的命令。

我们一直在不停地寻找避免重复的方法。设计的重复、编码的重复、文档的重复，当然还有构建的重复。Maven 最大化地消除了构建的重复，抽象了构建生命周期，并且为绝

大部分的构建任务提供了已实现的插件，我们不再需要定义过程，甚至不需要再去实现这些过程中的一些任务。最简单的例子是测试，我们没必要告诉 Maven 去测试，更不需要告诉 Maven 如何运行测试，只需要遵循 Maven 的约定编写好测试用例，当我们运行构建的时候，这些测试便会自动运行。

想象一下，Maven 抽象了一个完整的构建生命周期模型，这个模型吸取了大量其他的构建脚本和构建工具的优点，总结了大量项目的实际需求。如果遵循这个模型，可以避免很多不必要的错误，可以直接使用大量成熟的 Maven 插件来完成我们的任务（很多时候我们可能都不知道自己在使用 Maven 插件）。此外，如果有非常特殊的需求，我们也可以轻松实现自己的插件。

Maven 还有一个优点，它能帮助我们标准化构建过程。在 Maven 之前，十个项目可能有十种构建方式；有了 Maven 之后，所有项目的构建命令都是简单一致的，这极大地避免了不必要的学习成本，而且有利于促进项目团队的标准化。

综上所述，Maven 作为一个构建工具，不仅能帮我们自动化构建，还能够抽象构建过程，提供构建任务实现；它跨平台，对外提供了一致的操作接口，这一切足以使它成为优秀的、流行的构建工具。

1.1.3　Maven 不仅仅是构建工具

Java 不仅是一门编程语言，还是一个平台，通过 JRuby 和 Jython，我们可以在 Java 平台上编写和运行 Ruby 和 Python 程序。我们也应该认识到，Maven 不仅是构建工具，还是一个依赖管理工具和项目信息管理工具。它提供了中央仓库，能帮我们自动下载构件。

在这个开源的年代里，几乎任何 Java 应用都会借用一些第三方的开源类库，这些类库都可通过依赖的方式引入到项目中来。随着依赖的增多，版本不一致、版本冲突、依赖臃肿等问题都会接踵而来。手工解决这些问题是十分枯燥的，幸运的是 Maven 提供了一个优秀的解决方案，它通过一个坐标系统准确地定位每一个构件（artifact），也就是通过一组坐标 Maven 能够找到任何一个 Java 类库（如 jar 文件）。Maven 给这个类库世界引入了经纬，让它们变得有秩序，于是我们可以借助它来有序地管理依赖，轻松地解决那些繁杂的依赖问题。

Maven 还能帮助我们管理原本分散在项目中各个角落的项目信息，包括项目描述、开发者列表、版本控制系统地址、许可证、缺陷管理系统地址等。这些微小的变化看起来很琐碎，并不起眼，但却在不知不觉中为我们节省了大量寻找信息的时间。除了直接的项目信息，通过 Maven 自动生成的站点，以及一些已有的插件，我们还能够轻松获得项目文档、测试报告、静态分析报告、源码版本日志报告等非常具有价值的项目信息。

Maven 还为全世界的 Java 开发者提供了一个免费的中央仓库，在其中几乎可以找到任何的流行开源类库。通过一些 Maven 的衍生工具（如 Nexus），我们还能对其进行快速地搜索。只要定位了坐标，Maven 就能够帮我们自动下载，省去了手工劳动。

使用 Maven 还能享受一个额外的好处，即 Maven 对于项目目录结构、测试用例命名方

式等内容都有既定的规则，只要遵循了这些成熟的规则，用户在项目间切换的时候就免去了额外的学习成本，可以说是约定优于配置（Convention Over Configuration）。

1.2 为什么需要 Maven

Maven 不是 Java 领域唯一的构建管理的解决方案。本节将通过一些简单的例子解释 Maven 的必要性，并介绍其他构建解决方案，如 IDE、Make 和 Ant，并将它们与 Maven 进行比较。

1.2.1 组装 PC 和品牌 PC

笔者初中时开始接触计算机，到了高中时更是梦寐以求希望拥有一台自己的计算机。我的第一台计算机是赛扬 733 的，选购是一个漫长的过程，我先阅读了大量的杂志以了解各类配件的优劣，CPU、内存、主板、显卡，甚至声卡，我都仔细地挑选，后来还跑了很多商家，调货、讨价还价，组装好后自己装操作系统和驱动程序……虽然这花费了我大量时间，但我很享受这个过程。可是事实证明，装出来的机器稳定性不怎么好。

一年前我需要配一台工作站，这时候我已经没有太多时间去研究电脑配件了。我选择了某知名 PC 供应商的在线商店，大概浏览了一下主流的机型，选择了我需要的配置，然后下单、付款。接着 PC 供应商帮我组装电脑、安装操作系统和驱动程序。一周后，物流公司将电脑送到我的家里，我接上显示器、电源、鼠标和键盘就能直接使用了。这为我节省了大量时间，而且这台电脑十分稳定，商家在把电脑发送给我之前已经进行了很好的测试。对了，我还能享受两年的售后服务。

使用脚本建立高度自定义的构建系统就像买组装 PC，耗时费力，结果也不一定很好。当然，你可以享受从无到有的乐趣，但恐怕实际项目中无法给你那么多时间。使用 Maven 就像购买品牌 PC，省时省力，并能得到成熟的构建系统，还能得到来自于 Maven 社区的大量支持。唯一与购买品牌 PC 不同的是，Maven 是开源的，你无须为此付费。如果有兴趣，你还能去了解 Maven 是如何工作的，而我们无法知道那些 PC 巨头的商业秘密。

1.2.2 IDE 不是万能的

当然，我们无法否认优秀的 IDE 能大大提高开发效率。当前主流的 IDE 如 Eclipse 和 NetBeans 等都提供了强大的文本编辑、调试甚至重构功能。虽然使用简单的文本编辑器和命令行也能完成绝大部分开发工作，但很少有人愿意那样做。然而，IDE 是有其天生缺陷的：

- IDE 依赖大量的手工操作。编译、测试、代码生成等工作都是相互独立的，很难一键完成所有工作。手工劳动往往意味着低效，意味着容易出错。
- 很难在项目中统一所有的 IDE 配置，每个人都有自己的喜好。也正是由于这个原因，一个在机器 A 上可以成功运行的任务，到了机器 B 的 IDE 中可能就会失败。

我们应该合理利用 IDE，而不是过多地依赖它。对于构建这样的任务，在 IDE 中一次次地点击鼠标是愚蠢的行为。Maven 是这方面的专家，而且主流 IDE 都集成了 Maven，我们可以在 IDE 中方便地运行 Maven 执行构建。

1.2.3 Make

Make 也许是最早的构建工具，它由 Stuart Feldman 于 1977 年在 Bell 实验室创建。Stuart Feldman 也因此于 2003 年获得了 ACM 国际计算机组织颁发的软件系统奖。目前 Make 有很多衍生实现，包括最流行的 GNU Make 和 BSD Make，还有 Windows 平台的 Microsoft nmake 等。

Make 由一个名为 Makefile 的脚本文件驱动，该文件使用 Make 自己定义的语法格式。其基本组成部分为一系列规则（Rules），而每一条规则又包括目标（Target）、依赖（Prerequisite）和命令（Command）。Makefile 的基本结构如下：

```
TARGET… : PREREQUISITE…
COMMAND
….
…
```

Make 通过一系列目标和依赖将整个构建过程串联起来，同时利用本地命令完成每个目标的实际行为。Make 的强大之处在于它可以利用所有系统的本地命令，尤其是 UNIX/Linux 系统，丰富的功能、强大的命令能够帮助 Make 快速高效地完成任务。

但是，Make 将自己和操作系统绑定在一起了。也就是说，使用 Make，就不能实现（至少很难）跨平台的构建，这对于 Java 来说是非常不友好的。此外，Makefile 的语法也成问题，很多人抱怨 Make 构建失败的原因往往是一个难以发现的空格或 Tab 使用错误。

1.2.4 Ant

Ant 不是指蚂蚁，而是意指“另一个整洁的工具”（Another Neat Tool），它最早用来构建著名的 Tomcat，其作者 James Duncan Davidson 创作它的动机就是因为受不了 Makefile 的语法格式。我们可以将 Ant 看成是一个 Java 版本的 Make，也正因为使用了 Java，Ant 是跨平台的。此外，Ant 使用 XML 定义构建脚本，相对于 Makefile 来说，这也更加友好。

与 Make 类似，Ant 有一个构建脚本 build. xml，如下所示：

```
<?xml version = "1.0"?>
<project name = "Hello" default = "compile">
    <target name = "compile" description = "compile the Java source code to class
files">
        <mkdir dir = "classes"/>
        <javac srcdir = "." destdir = "classes"/>
    </target>
    <target name = "jar" depends = "compile" description = "create a Jar file ">
        <jar destfile = "hello.jar">
            <fileset dir = "classes" includes = "**/*.class"/>
            <manifest>
                <attribute name = "Main-Class" value = "HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

build. xml 的基本结构也是目标（target）、依赖（depends），以及实现目标的任务。比

如在上面的脚本中，jar 目标用来创建应用程序 jar 文件，该目标依赖于 compile 目标，后者执行的任务是创建一个名为 classes 的文件夹，编译当前目录的 java 文件至 classes 目录。compile 目标完成后，jar 目标再执行自己的任务。Ant 有大量内置的用 Java 实现的任务，这保证了其跨平台的特质，同时，Ant 也有特殊的任务 exec 来执行本地命令。

和 Make 一样，Ant 也都是过程式的，开发者显式地指定每一个目标，以及完成该目标所需要执行的任务。针对每一个项目，开发者都需要重新编写这一过程，这里其实隐含着很大的重复。Maven 是声明式的，项目构建过程和过程各个阶段所需的工作都由插件实现，并且大部分插件都是现成的，开发者只需要声明项目的基本元素，Maven 就执行内置的、完整的构建过程。这在很大程度上消除了重复。

Ant 是没有依赖管理的，所以很长一段时间 Ant 用户都不得不手工管理依赖，这是一个令人头疼的问题。幸运的是，Ant 用户现在可以借助 Ivy 管理依赖。而对于 Maven 用户来说，依赖管理是理所当然的，Maven 不仅内置了依赖管理，更有一个可能拥有全世界最多 Java 开源软件包的中央仓库，Maven 用户无须进行任何配置就可以直接享用。

1.2.5 不重复发明轮子[⊖]

小张是一家小型民营软件公司的程序员，他所在的公司要开发一个新的 Web 项目。经过协商，决定使用 Spring、iBatis 和 Tapstry。jar 包去哪里找呢？公司里估计没有人能把 Spring、iBatis 和 Tapstry 所使用的 jar 包一个不少地找出来。大家的做法是，先到 Spring 的站点上去找一个 spring-with-dependencies，然后去 iBatis 的网站上把所有列出来的 jar 包下载下来，对 Tapstry、Apache commons 等执行同样的操作。项目还没有开始，WEB-INF/lib 下已经有近百个 jar 包了，带版本号的、不带版本号的、有用的、没用的、相冲突的，怎一个“乱”字了得！

在项目开发过程中，小张不时地发现版本错误和版本冲突问题，他只能硬着头皮逐一解决。项目开发到一半，经理发现最终部署的应用的体积实在太大了，要求小张去掉一些没用的 jar 包，于是小张只能加班加点地一个个删……

小张隐隐地觉得这些依赖需要一个框架或者系统来进行管理。

小张喜欢学习流行的技术，前几年 Ant 十分流行，他学了，并成为了公司这方面的专家。小张知道，Ant 打包，无非就是创建目录，复制文件，编译源代码，使用一堆任务，如 copydir、fileset、classpath、ref、target，然后再 jar、zip、war，打包就成功了。

项目经理发话了：“兄弟们，新项目来了，小张，你来写 Ant 脚本！”

“是，保证完成任务！”接着，小张继续创建一个新的 XML 文件。`target clean; target compile; target jar;` …… 不知道他是否想过，在他写的这么多的 Ant 脚本中，有多少是重复劳动，有多少代码会在一个又一个项目中重现。既然都差不多，有些甚至完全相同，为什么每次都要重新编写？

⊖ 该小节内容整理自网友 Arthas 最早在 Maven 中文 MSN 群中的讨论，在此表示感谢

终于有一天，小张意识到了这个问题，想复用 Ant 脚本，于是在开会时他说：“以后就都用我这个规范的 Ant 脚本吧，新的项目只要遵循我定义的目录结构就可以了。”经理听后觉得很有道理：“嗯，确实是个进步。”

这时新来的研究生发言了：“经理，用 Maven 吧，这个在开源社区很流行，比 Ant 更方便。”小张一听很惊讶，Maven 真比自己的“规范化 Ant”强大？其实他不知道自己只是在重新发明轮子，Maven 已经有一大把现成的插件，全世界都在用，你自己不用写任何代码！

为什么没有人说“我自己写的代码最灵活，所以我不用 Spring，我自己实现 IoC；我不用 Hibernate，我自己封装 JDBC”？

1.3 Maven 与极限编程

极限编程（XP）是近些年在软件行业红得发紫的敏捷开发方法，它强调拥抱变化。该软件开发方法的创始人 Kent Beck 提出了 XP 所追求的价值、实施原则和推荐实践。下面看一下 Maven 是如何适应 XP 的。

首先看一下 Maven 如何帮助 XP 团队实现一些核心价值：

- **简单**。Maven 暴露了一组一致、简洁的操作接口，能帮助团队成员从原来的高度自定义的、复杂的构建系统中解脱出来，使用 Maven 现有的成熟的、稳定的组件也能简化构建系统的复杂度。
- **交流与反馈**。与版本控制系统结合后，所有人都能执行最新的构建并快速得到反馈。此外，自动生成的项目报告也能帮助成员了解项目的状态，促进团队的交流。

此外，Maven 更能无缝地支持或者融入到一些主要的 XP 实践中：

- **测试驱动开发**（TDD）。TDD 强调测试先行，所有产品都应该由测试用例覆盖。而测试是 Maven 生命周期的最重要的组成部分之一，并且 Maven 有现成的成熟插件支持业界流行的测试框架，如 JUnit 和 TestNG。
- **十分钟构建**。十分钟构建强调我们能够随时快速地从源码构建出最终的产品。这正是 Maven 所擅长的，只需要一些配置，之后用一条简单的命令就能让 Maven 帮你清理、编译、测试、打包、部署，然后得到最终的产品。
- **持续集成**（CI）。CI 强调项目以很短的周期（如 15 分钟）集成最新的代码。实际上，CI 的前提是源码管理系统和构建系统。目前业界流行的 CI 服务器如 Hudson 和 CruiseControl 都能很好地和 Maven 进行集成。也就是说，使用 Maven 后，持续集成会变得更加方便。
- **富有信息的工作区**。这条实践强调开发者能够快速方便地了解到项目的最新状态。当然，Maven 并不会帮你把测试覆盖率报告贴到墙上，也不会在你的工作台上放个鸭子告诉你构建失败了。不过使用 Maven 发布的项目报告站点，并配置你需要的项目报告，如测试覆盖率报告，都能帮你把信息推送到开发者眼前。

上述这些实践并非只在 XP 中适用。事实上，除了其他敏捷开发方法如 SCRUM 之外，

几乎任何软件开发方法都能借鉴这些实践。也就是说，Maven 几乎能够很好地支持任何软件开发方法。

例如，在传统的瀑布模型开发中，项目依次要经历需求开发、分析、设计、编码、测试和集成发布阶段。从设计和编码阶段开始，就可以使用 Maven 来建立项目的构建系统。在设计阶段，也完全可以针对设计开发测试用例，然后再编写代码来满足这些测试用例。然而，有了自动化构建系统，我们可以节省很多手动的测试时间。此外，尽早地使用构建系统集成团队的代码，对项目也是百利而无一害。最后，Maven 还能帮助我们快速地发布项目。

1.4 被误解的 Maven

C ++ 之父 Bjarne Stroustrup 说过一句话："只有两类计算机语言，一类语言天天被人骂，还有一类没人用。"当然这话也不全对，大红大紫的 Ruby 不仅有人用，而且骂的人也少。用户最多的 Java 得到的骂声就不绝于耳了。Maven 的用户也不少，它的邮件列表目前在 Apache项目中排名第 4（http://www.nabble.com/Apache-f90.html）。

让我们看看 Maven 受到了哪些质疑，笔者将对这些质疑逐一解释。

"Maven 对于 IDE（如 Eclipse 和 IDEA）的支持较差，bug 多，而且不稳定。"

相对于 JUnit 和 Ant 来说，Maven 比较年轻，IDE 集成等衍生产品还不够全面和成熟。但是，我们一定要知道，使用 Maven 最高效的方式永远是命令行，IDE 在自动化构建方面有天生的缺陷。此外，Eclipse 的 Maven 插件——m2eclipse 是一个比较优秀和成熟的工具，NetBeans 也在积极地为更好地集成 Maven 而努力，自 IntelliJ IDEA 开源后，也有望看到其对 Maven 更好的集成。

"Maven 采用了一个糟糕的插件系统来执行构建，新的、破损的插件会让你的构建莫名其妙地失败。"

自 Maven 2.0.9 开始，所有核心的插件都设定了稳定版本，这意味着日常使用 Maven 时几乎不会受到不稳定插件的影响。此外，Maven 社区也提倡为你使用的任何插件设定稳定的版本。如果我们有好的实践不采纳，遇到了问题就抱怨，未免不够公允。从 Maven 3 开始，如果你使用插件时未设定版本，会看到警告信息。

"Maven 过于复杂，它就是构建系统的 EJB 2。"

不要指望 Maven 十分简单，这几乎是不可能的。Maven 是用来管理项目的，清理、编译、测试、打包、发布，以及一些自定义的过程本身就是一件复杂的事情。目前在 Java 社区还有比 Maven 更强大、更简单的构建工具吗？答案是否定的。我们可以尝试去帮助 Maven 让它变得更简单，而不是抛弃它，然后自己实现一套更加复杂的构建系统。

"Maven 的仓库十分混乱，当无法从仓库中得到需要的类库时，我需要手工下载复制到本地仓库中。"

Maven 的中央仓库确实不完美，你也许会发现某个 jar 包出现在两个不同的路径下。这不

是 Maven 的错，这是开源项目本身改变了自身的坐标。如果没有中央仓库，你将不得不去开源项目首页寻找下载链接，这不是更费事吗？现在有很多的 Maven 仓库搜索服务。无法从中央仓库找到你需要的类库？由于许可证等因素，这是完全有可能的，这时你需要做的是建立一个组织内部的仓库服务器，你会发现这会给你带来许多意想不到的好处。

“缺乏文档是理解和使用 Maven 的一个主要障碍！”

这是事实。Maven 官方站点的文档十分凌乱，各种插件的文档更是需要费力寻找。Sonatype 编写的《Maven 权威指南》很好地改善了这一状况，但由于该书的某些部分与国内的现状有些脱离，且翻译速度无法跟上原版的更新速度，于是笔者编写本书，目的也是帮助大家理解和使用 Maven。

1.5　小结

本章只是从概念上简单地介绍了一下 Maven，通过本章我们应该能大致了解 Maven 是什么，以及它有什么用途。我们还将 Maven 与其他流行的构建工具（如 Make 和 Ant）做了一些比较和分析。如果你没用过 Maven，但有 Make 或者 Ant 的使用经验，相信通过比较你能更清楚地了解各种工具的优劣，并且会对 Maven 有一个理性的认识。

将 Maven 和极限编程结合起来分析是为了让大家从另一个角度了解 Maven，毕竟软件开发离不开对于软件过程的理解。

本章最后还收集了一些用户对 Maven 的误解，并逐条进行了分析和解释，希望能够消除大家的误解，从而积极地接受 Maven，最终从 Maven 中受益。