THE EXPERT'S VOICE® IN JAVA

Introducing Maven

Balaji Varanasi and Sudha Belida

Apress®

Contents at a Glance

Introduction

Introducing Maven provides a concise introduction to Maven, the de facto standard for building, managing, and automating Java and JEE-based projects in enterprises throughout the world. The book starts by explaining the fundamental concepts of Maven and showing you how to set up and test Maven on your local machine. It then delves deeply into concepts such as dependency management, life cycle phases, plug-ins, and goals. It also discusses project structure conventions, jump-starting project creation using archetypes, and documentation and report generation. Finally, it concludes with a discussion of Nexus and Maven's release process.

How This Book Is Structured

Chapter 1 starts with a gentle introduction to Maven. It discusses reasons for adopting Maven, and it provides an overview of its two alternatives: Ant and Gradle.

Chapter 2 focuses on setting up Maven on your machine and testing the installation. It also provides an overview of Maven's `settings.xml` file, and it shows you how to run Maven in a HTTP proxy-enabled environment.

Chapter 3 delves deeply into Maven's dependency management. It then discusses the GAV coordinates Maven uses for uniquely identifying its artifacts. Finally, it covers transitive dependencies and the impact they have on builds.

Chapter 4 discusses the organization of a basic Maven project and covers the important elements of a `pom.xml` file. Then you learn about testing the project using JUnit.

Chapter 5 provides detailed coverage of Maven's life cycle, plug-ins, build phases, and goals. It then walks you through the process of creating and using a simple Maven plug-in.

Chapter 6 introduces archetypes' project templates that enable you to bootstrap new projects quickly. The built-in archetypes are used to generate a Java project, a web project, and a multimodule project. You will then create a custom archetype from scratch and use it to generate a new project.

Chapter 7 covers the basics of site generation using Maven. It then discusses report generation and documentation such as Javadocs, test coverage reports, and FindBugs reports and how to integrate them into a Maven site.

Chapter 8 begins with a discussion of the Nexus repository manager and shows you how it can be integrated with Maven. It then provides complete coverage of Maven's release process and its different phases.

Target Audience

Introducing Maven is intended for developers and automation engineers who would like to get started quickly with Apache Maven. This book assumes basic knowledge of Java. No prior experience with Maven is required.

Downloading the Source Code

The source code for the examples in this book can be downloaded from `www.apress.com/9781484208427`. The source code is also available on GitHub at `https://github.com/bava/gswm-book`.

Once downloaded, unzip the code and place the contents in the `C:\apress\gswm-book folder`. The source code is organized by individual chapters. Where applicable, the chapter folders contain the `gswm` project with the bare minimum files to get you started on that chapter's code listings. The chapter folders also contain a folder named `final`, which holds the expected end state of the project(s).

Questions

We welcome reader feedback. If you have any questions or suggestions, you can contact the authors at `Balaji@inflinx.com` or `Sudha@inflinx.com`.

CHAPTER 1

Getting Started with Maven

Like other craftsmen, software developers rely on their tools to build applications. Developer's integrated development environments (IDEs), bug-tracking tools, build tools, frameworks, and debug tools, such as memory analyzers, play a vital role in day-to-day development and maintenance of quality software. This book will discuss and explore the features of Maven, which we know will become an important tool in your software development arsenal.

Apache Maven is an open source, standards-based project management framework that simplifies the building, testing, reporting, and packaging of projects. Maven's initial roots were in the Apache Jakarta Alexandria project that took place in early 2000. It was subsequently used in the Apache Turbine project. Like many other Apache projects at that time, the Turbine project had several subprojects, each with its own Ant-based build system. Back then, there was a strong desire for developing a standard way to build projects and to share generated artifacts easily across projects. This desire gave birth to Maven. Maven version 1.0 was released in 2004, followed by version 2.0 in 2005. At the time of writing this book, 3.0.5 is the current version of Maven.

Maven has become one of the most widely used open source software programs in enterprises around the world. Let's look at some of the reasons why Maven is so popular.

Standardized Directory Structure

Often, when we start work on a new project, a considerable amount of time is spent deciding on the project layout and folder structure needed to store code and configuration files. These decisions can vary vastly across projects and teams, which can make it difficult for new developers to understand and adopt other teams' projects. It can also make it hard for existing developers to jump between projects and find what they are seeking.

Maven addresses the above problems by standardizing the folder structure and organization of a project. Maven provides recommendations on where different parts of a project, such as source code, test code, and configuration files, should reside. For example, Maven suggests that all of the Java source code should be placed in the `src\main\java` folder. This makes it easier to understand and navigate any Maven project.

Additionally, these conventions make it easy to switch to and start using a new IDE. Historically, IDEs varied with project structure and folder names. A dynamic web project in Eclipse might use the WebContent folder to store web assets, whereas NetBeans might use Web Pages for the same purpose. With Maven, your projects follow a consistent structure and become IDE agnostic.

Declarative Dependency Management

Most Java projects rely on other projects and open source frameworks to function properly. It can be cumbersome to download these *dependencies* manually and keep track of their versions as you use them in your project.

Maven provides a convenient way to declare these project dependencies in a separate, external `pom.xml` file. It then automatically downloads those dependencies and allows you to use them in your project. This simplifies project dependency management greatly. It is important to note that in the `pom.xml` file you specify the *what* and not the *how*. The `pom.xml` file can also serve as a documentation tool, conveying your project dependencies and their versions.

Plug-ins

Maven follows a *plug-in–based architecture*, making it easy to augment and customize its functionality. These plug-ins encapsulate reusable build and task logic. Today, there are hundreds of Maven plug-ins available that can be used to carry out tasks ranging from code compilation to packaging to project documentation generation.

Maven also makes it easy to create your own plug-ins, thereby enabling you to integrate tasks and workflows that are specific to your organization.

Uniform Build Abstraction

Maven provides a uniform interface for building projects. You can build a Maven project by using just a handful of commands. Once you become familiar with Maven's build process, you can easily figure out how to build other Maven projects. This frees developers from having to learn build idiosyncrasies so they can focus more on development.

Tools Support

Maven provides a powerful command-line interface to carry out different operations. All major IDEs today provide excellent tool support for Maven. Additionally, Maven is fully integrated with today's continuous integration products such as Jenkins, Bamboo, and Hudson.

Archetypes

As we already mentioned, Maven provides a standard directory layout for its projects. When the time comes to create a new Maven project, you need to build each directory manually, and this can easily become tedious. This is where Maven archetypes come to rescue. *Maven archetypes* are predefined project templates that can be used to generate new projects. Projects created using archetypes will contain all of the folders and files needed to get you going.

Archetypes is also a valuable tool for bundling best practices and common assets that you will need in each of your projects. Consider a team that works heavily on Spring framework-based web applications. All Spring-based web projects share common dependencies and require a set of Spring configuration files. It is also highly possible that all of these web projects have similar `Log4j/Logback` configuration files, `CSS/Images`, and Apache Tile layouts or SiteMesh decorators. Maven lets this team bundle these common assets into an archetype. When new projects get created using this archetype, they will automatically have the common assets included. No more copy and pastes or drag and drops required.

Open Source

Maven is *open source* and costs nothing to download and use. It comes with rich online documentation and the support of an active community. Additionally, companies such as Sonatype offer commercial support for the Maven ecosystem.

CONVENTION OVER CONFIGURATION

Convention over configuration (CoC) or *coding by convention* is one of the key tenants of Maven. Popularized by the Ruby on Rails community, CoC emphasizes sensible defaults, thereby reducing the number of decisions to be made. It saves time and also results in a simpler end product, as the amount of configuration required is drastically reduced.

As part of its CoC adherence, Maven provides several sensible defaults for its projects. It lays out a standard directory structure and provides defaults for the generated artifacts. Imagine looking at a Maven artifact with the name `log4j-1.4.3.jar`. At a glance, you can easily see that you are looking at a `log4j` JAR file, version 1.4.3.

One drawback of Maven's CoC is the rigidness that end users experience when using it. To address this, you can customize most of Maven's defaults. For example, it is possible to change the location of the Java source code in your project. As a rule of thumb, however, such changes to defaults should be minimized.

Maven Alternatives

Although the emphasis of this book is on Maven, let's look at a couple of its alternatives: Ant + Ivy and Gradle.

Ant + Ivy

Apache Ant (http://ant.apache.org) is a popular open source tool for scripting builds. Ant is Java based, and it uses Extensible Markup Language (XML) for its configuration. The default configuration file for Ant is the `build.xml` file.

Using Ant typically involves defining tasks and targets. As the name suggests, an *Ant task* is a unit of work that needs to be completed. Typical tasks involve creating a directory, running a test, compiling source code, building a web application archive (WAR) file, and so forth. A *target* is simply a set of tasks. It is possible for a target to depend on other targets. This dependency lets us sequence target execution. Listing 1-1 demonstrates a simple `build.xml` file with one target called *compile*. The `compile` target has two echo tasks and one `javac` task.

Listing 1-1. Sample Ant `build.xml` File

```
<project name="Sample Build File" default="compile" basedir=".">

   <target name="compile" description="Compile Source Code">
      <echo message="Starting Code Compilation"/>
      <javac srcdir="src" destdir="dist"/>
      <echo message="Completed Code Compilation"/>
   </target>

</project>
```

Ant doesn't impose any conventions or restrictions on your project and it is known to be extremely flexible. This flexibility has sometimes resulted in complex, hard-to-understand and maintain `build.xml` files.

Apache Ivy (http://ant.apache.org/ivy/) provides automated dependency management, making Ant more joyful to use. With Ivy, you declare the dependencies in an XML file called `ivy.xml`, as shown in Listing 1-2. Integrating Ivy with Ant involves declaring new targets in the `build.xml` file to retrieve and resolve dependencies.

Listing 1-2. Sample Ivy Listing

```
<ivy-module version="2.0">
  <info organisation="com.apress" module="gswm-ivy" />

  <dependencies>
        <dependency org="org.apache.logging.log4j" name="log4j-api" 
rev="2.0.2" />
  </dependencies>
</ivy-module>
```

Gradle

Gradle (http://gradle.org/) is the newest addition to the Java build project automation tool family. Unlike Ant and Maven, which use XML for configuration, Gradle uses a Groovy-based *Domain Specific Language* (DSL).

Gradle provides the flexibility of Ant, and it uses the same notion of tasks. It also follows Maven's conventions and dependency management style. Listing 1-3 shows a default `build.gradle` file.

Listing 1-3. Default `build.gradle` File

```
apply plugin: 'java'

version = '1.0'

repositories {
   mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.10'
}
```

Gradle's DSL and its adherence to CoC results in compact build files. The first line in Listing 1-3 includes a Java plug-in for `build`'s use. Plug-ins in Gradle provide preconfigured tasks and dependencies to the project. The Java plug-in, for example, provides tasks for building source files, running unit tests, and installing artifacts. The dependencies section in the `default.build` file instructs Gradle to use `JUnit` dependency during the compilation of test source files. Gradle's flexibility, like that of Ant, can be abused, which results in difficult and complex builds.

Summary

Apache Maven greatly simplifies the build process and automates project management tasks. This chapter provided a gentle introduction to Maven and described the main reasons for adopting it. We also looked at Maven's close peers: Ant + Ivy and Gradle.

In the next chapter, you will learn about the set up required to get up and running with Maven.

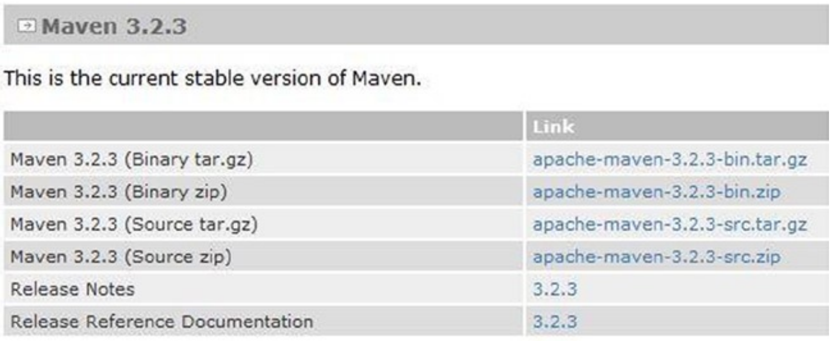
CHAPTER 2

Setting Up Maven

Maven installation is an easy and straightforward process. This chapter will explain how to install and set up Maven using the Windows 7 operating system. You can follow the same procedure with other operating systems.

Note Maven is a Java-based application and requires the Java Development Kit (JDK) to function properly. Maven version 3.2 requires JDK 1.6 or above and versions 3.0/3.1 can be run using JDK 1.5 or above. Before proceeding with Maven installation, make sure that you have Java installed. If not, install the JDK (not just Java Runtime Environment [JRE]) from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. In this book, we will be using JDK 1.7.

You will begin the installation process by downloading the latest version of Maven from the Apache Maven web site (`http://maven.apache.org/download.html`). At the time of this writing, the latest version is 3.2.3. Download the Maven 3.2.3 binary `.zip` file as shown in Figure 2-1.



Maven 3.2.3

This is the current stable version of Maven.

	Link
Maven 3.2.3 (Binary tar.gz)	apache-maven-3.2.3-bin.tar.gz
Maven 3.2.3 (Binary zip)	apache-maven-3.2.3-bin.zip
Maven 3.2.3 (Source tar.gz)	apache-maven-3.2.3-src.tar.gz
Maven 3.2.3 (Source zip)	apache-maven-3.2.3-src.zip
Release Notes	3.2.3
Release Reference Documentation	3.2.3

Figure 2-1. *Maven download page*