

THE EXPERT'S VOICE® IN JAVA

Introducing Maven

Balaji Varanasi and Sudha Belida

Apress®

Для вашего удобства Apress разместил некоторые вводные материалы сразу после алфавитного указателя. Пожалуйста, используйте закладки и ссылки из Краткого содержания для доступа к ним.



Краткое содержание

Введение	4
Глава 1: Начало работы с Maven	6
Глава 2: Установка Maven	12
Глава 3: Управление зависимостями Maven	19
Глава 4: Основы Maven-проекта	27
Глава 5: Жизненный цикл Maven	40
Глава 6: Архетипы Maven	48
Глава 7: Документация и отчетность	64
Глава 8: Публикация с помощью Maven	77
Предметный указатель	97
Introducing Maven	101
Об авторах	107
О техническом редакторе	108
Благодарности	109

Введение

Введение в Maven представляет собой краткое введение в Maven, де-факто являющегося промышленным стандартом для построения, управления и автоматизации Java и JEE-проектов по всему миру. Книга начинается с объяснения фундаментальных концепций Maven, наглядно демонстрируя, как настроить и протестировать Maven на локальном компьютере. Затем она глубоко углубляется в такие понятия, как управление зависимостями, этапы жизненного цикла, плагины и цели. Также обсуждаются соглашения о структуре проектов, быстрое создание проектов с использованием архетипов, а также документирование и создание отчетов. В итоге книга завершается обсуждением процесса выпуска проектов Nexus и Maven.

Как построена эта книга

Глава 1 постепенно знакомит вас с **Maven**. Обсуждаются причины появления **Maven** и дается обзор двух альтернатив – **Ant** и **Gradle**.

Глава 2 посвящена настройке **Maven** на локальной машине и проверке установки. Также дается обзор файла **Maven settings.xml** и демонстрируется, как запустить **Maven** в среде с разрешенным прокси HTTP.

Глава 3 сначала глубоко погружается в управление зависимостями **Maven**. Затем обсуждаются **GAV**-координаты, которые **Maven** использует для однозначной идентификации своих артефактов. В завершение рассматриваются транзитивные зависимости и влияние, которое они оказывают на сборку.

Глава 4 обсуждается организация основы **Maven**-проекта и рассматриваются важные элементы файла **pom.xml**. Затем вы узнаете о тестировании проекта с использованием **JUnit**.

Глава 5 дает глубокий охват жизненного цикла **Maven**, плагинов, фаз сборки и целей. Затем она проведет вас через процесс создания и использования простого плагина **Maven**.

Глава 6 ознакомит с шаблонами проекта архетипов, которые позволяют быстро запускать новые проекты. Встроенные архетипы используются для создания **Java**-проекта, веб-проекта и многомодульного проекта. Затем вы с нуля создадите пользовательский архетип и используете его для генерации нового проекта.

Глава 7 посвящена основам создания сайта с использованием **Maven**. Далее обсуждается вопрос создания отчетов и документации с помощью **Javadoc**, отчеты о степени покрытия тестами, отчеты **FindBugs** и как их интегрировать в сайт, построенный на **Maven**.

Глава 8 начинается с обсуждения менеджера репозитория **Nexus** и демонстрация того, как он может быть интегрирован с **Maven**. Затем дается полный обзор процесса выпуска релиза с помощью **Maven** и его различных фаз.

Для кого эта книга

Введение в Maven предназначена для разработчиков и инженеров по автоматизации, которые хотели бы быстро начать работу с **Apache Maven**. Эта книга предполагает базовые знания **Java**. Опыт работы с **Maven** не требуется.

Получение исходного кода

Исходный код для примеров этой книги может быть загружен с www.apress.com/9781484208427. Также исходный код доступен на **GitHub** по ссылке <https://github.com/bava/gswm-book>.

После скачивания распакуйте архив и разместите его содержимое в папке `C:\apress\gswm-book`. Исходный код распределен по отдельным главам. Там, где это требуется, папки глав содержат проект **gswm** с минимальным набором файлов, позволяющий просматривать код главы. Папки глав также содержат подпапку с именем **final**, которая содержит проект в завершённом состоянии.

Вопросы

Мы приветствуем обратную связь с читателями. Если у вас имеются какие-либо вопросы или предложения, вы можете связаться с авторами по адресам электронной почты balaji@inflinx.com или sudha@inflinx.com.

Глава 1: Начало работы с Maven

Как и всякие мастера, разработчики программного обеспечения при создании приложений полагаются на свои инструменты. Интегрированные среды разработки (**IDE**), инструменты отслеживания ошибок, инструменты для сборки, фреймворки, инструменты отладки такие, как анализаторы памяти, играют важную роль в повседневном развитии и поддержке качественных программ. Эта книга посвящена обсуждению и раскрытию возможностей **Maven**, который, мы в этом уверены, станет важным инструментом в вашем арсенале разработчика.

Apache Maven - это открытый фреймворк, построенный на стандартах разработки проектов, который упрощает сборку, тестирование, отчетность и запаковку проектов. Истоки **Maven** лежат в проекте **Apache Jakarta Alexandria**, который развивался в начале 2000-х годов. Впоследствии он использовался в проекте **Apache Turbine**. Подобно многим другим **Apache**-проектам того времени проект **Turbine** имел несколько дочерних проектов, каждый из которых обладал собственной системой сборки, основанной на **Ant**. Существовала сильная потребность для возникновения стандартного способа построения проектов и простого способа передачи сущностей между проектами. Эта потребность и привела к появлению **Maven**. Версия **Maven 1.0** была выпущена в 2004 году, а последовавшая за ней версия **2.0** - в 2005-м. На момент написания этой книги текущей версией **Maven** является версия **3.0.5**.

Maven стал одним из наиболее распространенных открытых программных продуктов в индустрии по всему миру. Давайте рассмотрим некоторые причины, по которым **Maven** оказался настолько популярным.

Стандартная структура папок

Часто, когда мы начинаем работу над новым проектом, значительное количество времени тратится на принятии решения о планировке проекта и структуре папок, необходимых для хранения кода и конфигурационных файлов. Эти решения могут значительно отличаться у разных проектов и команд, что может осложнить новым сотрудникам понимание и восприятие проектов других команд. Это также может затруднить действующим разработчикам переключение между проектами и поиск интересующей их информации.

Maven решает эти проблемы путем стандартизации структуры папок и внутреннего устройства проекта. **Maven** дает рекомендации, где должны находиться различные части проекта, такие, как исходный код, код тестов и конфигурационные файлы. Например, **Maven** предполагает, что весь исходный код **Java** должен быть размещен в папке `src/main/java`. Всё это облегчает понимание и навигацию по любому проекту **Maven**.

Кроме того, эти соглашения облегчают переход и начало использования новой **IDE**. Исторически у разных **IDE** структура проекта и имена папок различаются. Динамические веб-проекты в **Eclipse** могут использовать папку `WebContent` для хранения ресурсов проекта, в то время, как **NetBeans** для тех же целей

может использовать папку с именем **WebPages**. С помощью **Maven** ваши проекты будут придерживаться определенной структуры и перестанут зависеть от **IDE**.

Декларативное управление зависимостями

Большинство проектов **Java** для правильного функционирования полагаются на другие проекты и фреймворки с открытым кодом. Ручное скачивание этих зависимостей и поддержка их версий при работе над проектом может оказаться довольно затруднительным делом.

Maven предоставляет удобный способ объявить зависимости проекта в отдельном, внешнем файле с именем `pom.xml`. После чего **Maven** автоматически загружает эти зависимости и позволяет вам использовать их в своем проекте. Это значительно упрощает управление зависимостями проекта. Важно отметить, что в файле `pom.xml` вы указываете, что зависит, а не как. Кроме того, файл `pom.xml` выступает в качестве инструмента документирования, сообщая о зависимостях вашего проекта и его версии.

Плагины

Maven использует архитектуру, основанную на плагинах, что делает его легко расширяемым, а его функциональность – легко настраиваемой. Эти плагины инкапсулируют многократно используемые алгоритмы сборки и выполнения задач. Сегодня существуют сотни доступных **Maven**-плагинов, которые могут быть использованы для выполнения широкого круга задач, от компиляции кода до генерации проектной документации.

Кроме того, **Maven** позволяет легко создавать свои собственные плагины, тем самым позволяя вам выполнять задачи и процессы, специфичные для вашей организации.

Единая абстракция сборки

Maven обеспечивает единый интерфейс для сборки проектов. Вы можете построить **Maven**-проект используя всего несколько команд. После того, как вы познакомитесь с процессом сборки **Maven**, вы без труда поймете, как строить и другие проекты **Maven**. Это освобождает разработчиков от необходимости изучать индивидуальные особенности построения, позволяя таким образом сосредоточиться на разработке.

Поддержка инструментов

Maven предоставляет мощный инструмент командной строки выполнения различных операций. Сегодня все основные **IDE** предоставляют прекрасную поддержку инструментов для **Maven**. В добавок, **Maven** полностью интегрирован с такими современными инструментами непрерывной разработки, как **Jenkins**, **Bamboo**, и **Hudson**.

Архетипы

Как мы уже упоминали, **Maven** обеспечивает стандартное расположение папок для своих проектов. Когда приходит время создавать новый проект **Maven**, вам приходится вручную создавать каждую папку, что может быстро стать утомительным. И это тот самый момент, когда архетипы **Maven** приходят на помощь. **Архетипы Maven** являются предопределенными шаблонами проектов, которые могут быть использованы для создания новых проектов. Проекты, созданные с помощью архетипов, будут содержать все папки и файлы, необходимые для работы.

Кроме того, архетипы являются ценным инструментом для хранения лучших методик и совместных активов, которые будут нужны в каждом проекте.

Архетипы также являются ценным инструментом для обобщения лучших практик и общих активов, которые вам понадобятся в каждом из ваших проектов. Представим себе команду, которая в основном работает над веб-приложениями, основанными на фреймворке **Spring**. Все веб-проекты на **Spring** используют общие зависимости и требуют схожего набора конфигурационных файлов **Spring**. Также весьма вероятно, что все эти веб-проекты будут иметь одинаковые конфигурационные файлы **Log4j/Logback**, **CSS**, изображений, шаблонов **Apache Tile** или декораторов **SiteMesh**. **Maven** позволит этой команде сгруппировать такие общеиспользуемые активы в единый архетип. И при создании новых проектов с использованием этого архетипа в них автоматически будут включены все общие активы. Никакого копирования и перетаскивания больше не понадобится.

Открытый исходный код

Maven является открытым проектом и бесплатен для скачивания и использования. Он поставляется с обширной онлайн-документацией и поддерживается активным сообществом. Кроме того, такие компании, как **Sonatype**, предлагают поддержку экосистем **Maven** на платной основе.

СОГЛАШЕНИЕ ПО КОНФИГУРАЦИИ

Соглашение по конфигурации (Convention Over Configuration, CoC) или кодирование по конвенции является одной из ключевых особенностей **Maven**. Ставшее известной благодаря сообществу **Ruby on Rails**, **CoC** основывается на важных умолчаниях, снижая количество решений, которые приходится принимать. Это экономит время и упрощает создание конечного продукта за счет радикального сокращения объема требуемого конфигурирования.

В рамках приверженности **CoC**, **Maven** обеспечивает несколько важных умолчаний, закладывая стандартную структуру папок и предоставляя значения по умолчанию для создаваемых сущностей. Например, взглянув на сущность с именем `log4j-1.4.3.jar` вы с первого взгляда легко поймете, что это JAR-файл библиотеки журналирования **log4j**, версия **1.4.3**.

Одним из недостатков **CoC Maven** является жесткость, с которой сталкиваются использующие её конечные пользователи. Для решения этой проблемы у вас имеется возможность изменить большинство настроек **Maven**. Например, можно изменить в проекте расположение исходных кодов **Java**. Но, как правило, такие изменения следует сводить к минимуму.

Альтернативы Maven

Невзирая на то, что эта книга сосредоточена на **Maven** давайте рассмотрим пару её альтернатив: **Ant + Ivy** и **Gradle**.

Ant + Ivy

Apache Ant (<http://ant.apache.org>) является популярным инструментом с открытым кодом для создания сценариев сборки. **Ant** основан на **Java**, и использует расширяемый язык разметки (**Extensible Markup Language, XML**) для конфигурирования. Конфигурация **Ant** по умолчанию содержится в файле [build.xml](#).

Использование **Ant** обычно заключается в определении задач и целей. Как явствует из названия, задачей в **Ant** является единица исполняемой работы. Обычные задачи состоят из создания папки, запуска теста, компиляции исходного кода, построения файла архива веб-приложения (**WAR**) и т.д. Цель **Ant** – это просто набор задач. Одна цель может зависеть от других целей. Такая зависимость дает нам возможность последовательного исполнения целей. **Листинг 1-1** демонстрирует простой файл [build.xml](#), содержащий одну цель с именем **compile**. Эта цель содержит две задачи для вывода сообщений **echo** и одну задачу запуска компилятора **javac**.

Листинг 1-1. Пример Ant-файла [build.xml](#)

```
<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 не накладывает каких-либо условий или ограничений на ваш проект и известен своей чрезвычайной гибкостью. Эта гибкость иногда приводит к появлению сложных, тяжелых в понимании и поддержке файлов [build.xml](#).

Apache Ivy (<http://ant.apache.org/ivy/>) предоставляет автоматизированное управление зависимостями, что делает использование **Ant** более интересным. С помощью **Ivy** вы объявляете зависимости в XML-файле, носящем имя [ivy.xml](#), как показано в **Листинге 1-2**. Интеграция **Ivy** с **Ant** заключается в объявлении новых целей в файле [build.xml](#) для получения и разрешения зависимостей.

Listing 1-2. Пример Ivy-файла `ivy.xml`

```
<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/>) является новейшим дополнением к семейству инструментов автоматизации процессов сборки **Java**-проектов. В отличие от **Ant** или **Maven**, использующих XML для конфигурирования, **Gradle** использует основанный на **Groovy** предметно-ориентированный язык **Domain Specific Language (DSL)**.

Gradle обеспечивает гибкость, как у **Ant**, и использует точно такое же представление задач. Кроме того, он следует соглашениям и стилю управления зависимостями, как у **Maven**. В **Листинге 1-3** показано содержимое файла `build.gradle` по умолчанию.

Listing 1-3. Содержимое файла `build.gradle` по умолчанию

```
apply plugin: 'java'

version = '1.0'

repositories {
  mavenCentral()
}

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

Предметно-ориентированный язык **DSL** системы сборки **Gradle** и то, что он следует принципам **CoC**, являются причиной компактности `gradle`-файлов сборки. Первая строка в **Листинге 1-3** содержит указание на плагин **Java**, используемый для построения. Плагины в **Gradle** обеспечивают проект предварительно сконфигурированными задачами и зависимостями. Плагин **Java**, например, обеспечивает задачи для сборки файлов исходного кода, запуска юнит-тестов и установки ресурсов. Секция **dependencies** файла указывают **Gradle** использовать зависимости **JUnit** при компиляции исходных файлов тестов. Гибкость **Gradle**, как и гибкость **Ant**, в случае злоупотребления может стать причиной возникновения тяжелых и сложных сборок.

Итоги

Apache Maven существенно упрощает процесс сборки проекта и автоматизирует задачи по управлению проектами. Эта глава постепенно знакомит вас с **Maven** и описывает основные причины для его использования. Мы также рассмотрели ближайшие альтернативы Maven: **Ant + Ivy** и **Gradle**.

В следующей главе вы узнаете о том, как установить и запустить **Maven**.

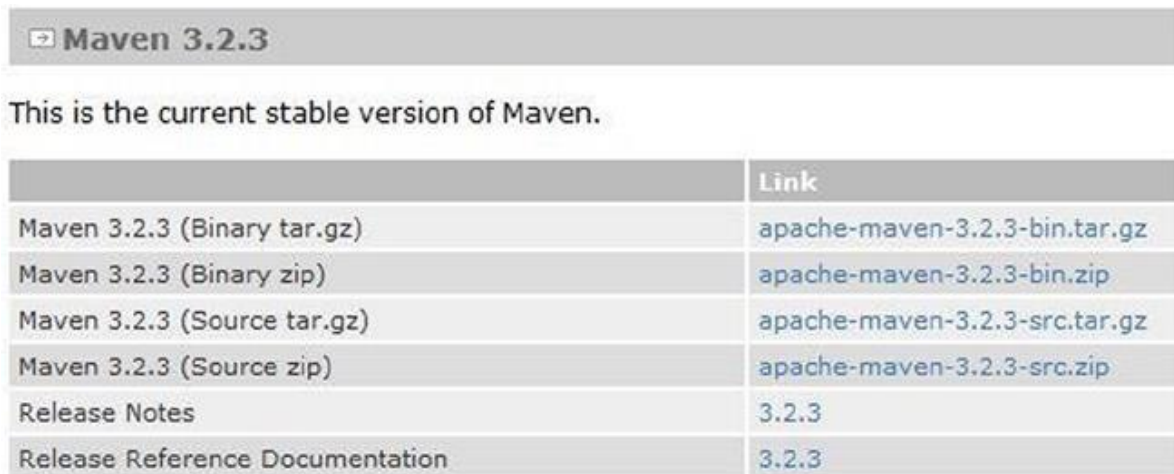
Глава 2: Установка Maven

Установка **Maven** является простым и прямолинейным процессом. В этой главе объясняется, как установить и настроить Maven для операционной системы **Windows 7**. Вы можете следовать той же процедуре и в других операционных системах.

Внимание. Maven является **Java**-приложением и для работы требует наличия **Java Development Kit (JDK)**. Maven версии **3.2** требует **JDK 1.6** или выше, а версии **3.0/3.1** могут быть запущены с использованием **JDK 1.5** или выше. Перед установкой **Maven** убедитесь, что у вас уже установлена **Java**. Если нет, то установите **JDK** (а не только среду исполнения **Java [JRE]**) из <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. В этой книге мы будем использовать **JDK 1.7**.

Вы начнете процесс установки со скачивания самой последней версии **Maven** с веб-сайта **Apache Maven** (<http://maven.apache.org/download.html>). В момент написания данной книги самой последней версией являлась **3.2.3**. Скачайте бинарный **zip**-файл **Maven 3.2.3**, показанный на **Рисунке 2-1**.

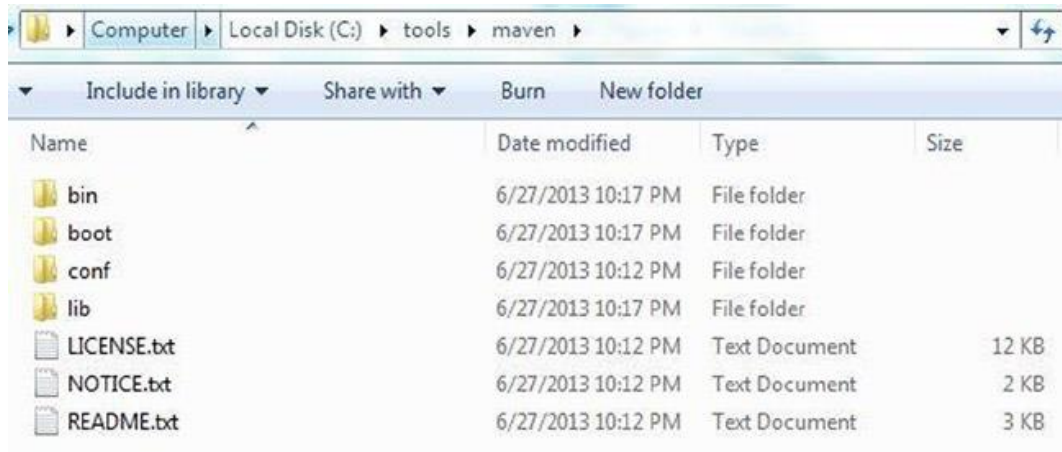
Рисунок 2-1. Страница скачивания **Maven**



	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

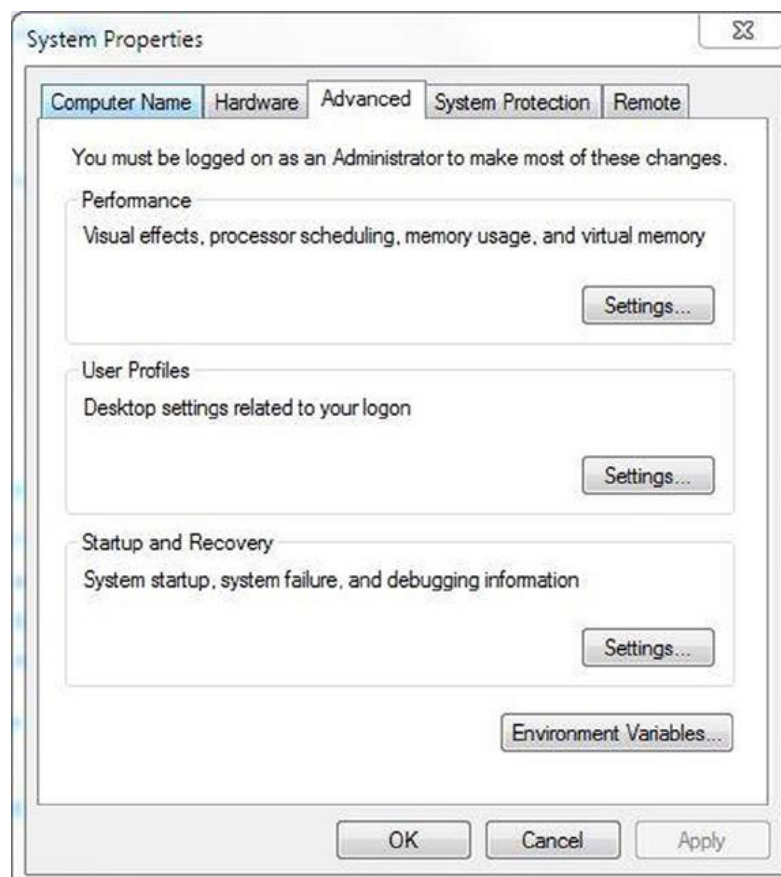
Когда скачивание завершится, распакуйте дистрибутив в локальную директорию на своем компьютере. В ней будет создана папка [apache-maven-3.2.3-bin](#). В этой книге предполагается, что вы разместили содержимое папки [apache-maven-3.2.3-bin](#) в директории `c:\tools\maven`, как показано на **Рис.2-2**.

Рисунок 2-2. Место установки **Maven**



Следующим этапом процесса установки является добавление переменной окружения **M2_HOME**, указывающей на директорию установки **Maven**, в нашем случае на `c:\tools\maven`. Откройте меню **Пуск** и нажмите правую кнопку мыши на пункте меню «**Компьютер**». Затем в открывшемся контекстном меню выберите пункт «**Свойства**» и после этого перейдите на закладку «**Дополнительные параметры системы**». Откроется окно, изображенное на **Рис. 2-3**.

Рисунок 2-3. Окно «Свойства системы»



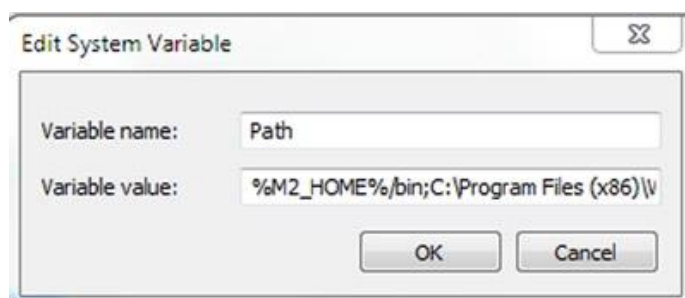
Нажмите кнопку «Переменные среды» (Environment Variables) и затем кнопку «Создать» (New) под окошком «Системные переменные» (System variables). Введите значения, показанные на Рис.2-4, и нажмите ОК.

Рисунок 2-4. Системная переменная M2_HOME



Последним шагом процесса является изменение переменной окружения **Path** таким образом, чтобы вы смогли вызывать команды **Maven** из командной строки. Выберите переменную **Path** и нажмите кнопку «Изменить» (Edit). Добавьте строку `%M2_HOME%/bin` в начало значения пути, как показано на Рис. 2-5 и нажмите ОК. На этом установка **Maven** завершается. Если у вас были открыты окна командной строки, закройте их и откройте новое окно ввода командной строки. Когда переменные окружения добавляются или изменяются, новые значения не распространяются автоматически на уже открытые окна командных строк.

Рисунок 2-5. Добавление M2_HOME в переменную Path



ПЕРЕМЕННАЯ ОКРУЖЕНИЯ MAVEN_OPTS

При использовании Maven, особенно в сложных проектах, вы можете столкнуться с ошибками типа **OutOfMemory**. Это происходит, например, когда вы запускаете большое количество **JUnit** тестов или, когда вы создаёте большое количество отчетов. Для устранения этой ошибки увеличьте объем памяти, выделяемый для виртуальной машины **Java (JVM)**, используемой **Maven**. Это делается глобально путем создания новой переменной среды с именем **MAVEN_OPTS**. Для начала мы рекомендуем использовать значение `-Xmx512m`.

Проверка установки

Теперь, когда **Maven** установлен, пришло время проверить установку. Откройте **Командную строку** и вызовите следующую команду:

```
mvn -v
```

Эта команда приведет к выводу примерно такой информации:

```
C:\Windows\System32>mvn -v
Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4;
2014-08-11T14:58:10-06:00)
Maven home: c:\tools\maven
Java version: 1.7.0_25, vendor: Oracle Corporation
Java home: C:\Java\jdk1.7.0_25\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "x86", family: "windows"
```

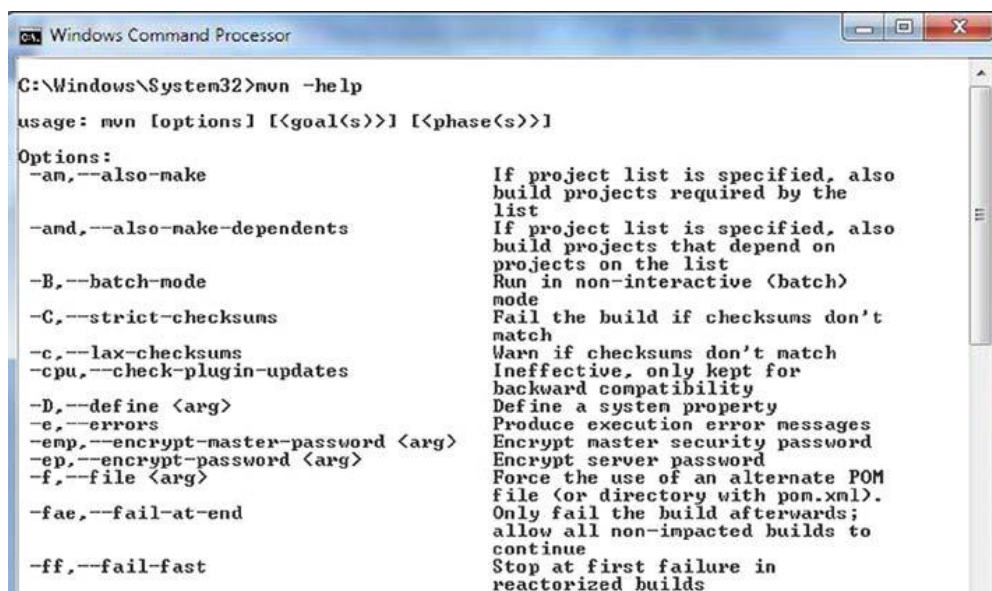
Опция командной строки **-v** сообщает путь, где установлен **Maven**, и какая версия **Java** используется. Вы бы получили тот же результат, если бы запустили расширенную команду **mvn --version**.

Получение помощи

Можно получить список параметров командной строки **Maven** путем использования ключей **--help** или **-h**. Запуск следующей команды выведет результат похожий на тот, что изображен на **Рис.2-6**.

```
mvn -h
```

Рисунок 2-6. Результат запуска команды **Maven --help**



```
Windows Command Processor
C:\Windows\System32>mvn -help
usage: mvn [options] [<goal(s)>] [<phase(s)>]
Options:
-am,--also-make                If project list is specified, also
                               build projects required by the
                               list
-and,--also-make-dependents    If project list is specified, also
                               build projects that depend on
                               projects on the list
-B,--batch-mode                Run in non-interactive (batch)
                               mode
-C,--strict-checksums          Fail the build if checksums don't
                               match
-c,--lax-checksums             Warn if checksums don't match
-cpu,--check-plugin-updates    Ineffective, only kept for
                               backward compatibility
-D,--define <arg>              Define a system property
-e,--errors                    Produce execution error messages
-emp,--encrypt-master-password <arg> Encrypt master security password
-ep,--encrypt-password <arg>   Encrypt server password
-f,--file <arg>                Force the use of an alternate POM
                               file (or directory with pom.xml).
-fae,--fail-at-end             Only fail the build afterwards;
                               allow all non-impacted builds to
                               continue
-ff,--fail-fast                Stop at first failure in
                               rectorized builds
```

Дополнительные настройки

Выполненных нами шагов по установке уже достаточно для начала работы с **Maven**. Однако, в большинстве случаев необходимо ввести дополнительную конфигурационную информацию. Конфигурация для конкретного пользователя указывается в файле `settings.xml`, расположенном в папке `c:\Users\<<user_name>>\.m2`.

Замечание: Папка `.m2` важна для правильной работы **Maven**. Помимо прочего эта папка содержит файл `settings.xml` и папку репозитория. Папка репозитория содержит `jar-файлы` плагинов и требуемые для **Maven** метаданные. Также она содержит связанные с проектом `jar-файлы`, которые **Maven** скачал из сети *Интернет*. Мы более подробно рассмотрим эту папку в **Главе 3**.

По умолчанию папка `.m2` расположена в директории пользователя. В **ОС Windows** это обычно папка `c:\Users\<<your_user_name>>`. **Maven** автоматически создает папку `.m2`. Но если у вас этой папки нет, то создайте её сами.

Сразу после установки папка `.m2` не содержит файла `settings.xml`. В папке `.m2` вашего локального компьютера создайте файл `settings.xml` и скопируйте в него структуру файла, приведенную в **Листинге 2-1**. Мы рассмотрим некоторые из этих элементов в следующих главах. Краткое описание этих элементов приведено в **Таблице 2-1**.

Листинг 2-1. Базовое содержимое шаблона `settings.xml`

```
<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">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```


Таблица 2-1. Описание элементов `settings.xml`

Элемент	Описание
localRepository	Maven хранит локальные копии плагинов и зависимостей в папке <code>c:\Users\<<your_user_name>>\.m2\repository</code> . Этот элемент может быть использован для изменения пути к этому локальному хранилищу. Например, <code><localRepository>c:\mavenrepo</localRepository></code> изменит расположение хранилища на папку <code>mavenrepo</code> .
interactiveMode	Как следует из названия, когда это значение установлено в true (являющееся значением по умолчанию), то Maven ожидает ввода от пользователя.
offline	Когда установлена в true , эта настройка указывает Maven работать в офлайн-режиме. Значение по умолчанию – false .
servers	Maven может взаимодействовать с большим количеством серверов, таких, как серверы Apache Subversion (SVN) , серверы сборки и серверами удалённых хранилищ. Этот элемент позволяет указывать специальные настройки безопасности, такие как имя и пароль пользователя, которые требуются для подключения к этим серверам.
mirrors	Как следует из названия, настройка позволяет указывать альтернативные расположения для хранилищ.
proxies	Содержит информацию об HTTP-прокси, необходимые для подключения к Интернет.
profiles	Профили позволяют группировать определенные элементы конфигурации, такие, как хранилища и pluginRepositories .
activeProfile	Позволяет указать профиль по умолчанию, который будет использоваться Maven.

Настройка прокси

Как будет подробно обсуждаться в **Главе 3, Maven** требует Интернет-соединения для скачивания плагинов и зависимостей. Некоторые компании используют HTTP-прокси для ограничения доступа в **Интернет**. В таких случаях запуск **Maven** приводит к ошибкам вида «**Unable to download artifact**». Для решения этой проблемы следует отредактировать файл `settings.xml` и добавить в него информацию о прокси вашей компании. Пример конфигурации показан в **Листинге 2-2**.

Листинг 2-2. Файл `settings.xml` с информацией о прокси

```
<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">
  <proxies>
    <proxy>
      <id>companyProxy</id>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.company.com</host>
      <port>8080</port>
      <username>proxyusername</username>
      <password>proxypassword</password>
      <nonProxyHosts/>
    </proxy>
  </proxies>
</settings>
```

Поддержка IDE

На протяжении этой книги мы будем использовать командную строку для создания и сборки примеров приложений. Если вы собираетесь использовать **IDE**, то хорошей новостью для вас будет то, что все современные **IDE** поставляются с полной поддержкой **Maven** без необходимости каких-либо дополнительных настроек.

Итоги

В этой главе мы рассматривали настройку **Maven** на локальной машине. Вы узнали, что **Maven** загружает плагины и сущности, необходимые для своей работы. Эти сущности сохраняются в папке `m2repository`. Папка `.m2` также содержит файл `settings.xml`, который может быть использован для настройки поведения **Maven**.

В следующей главе мы более детально рассмотрим управление зависимостями **Maven**.

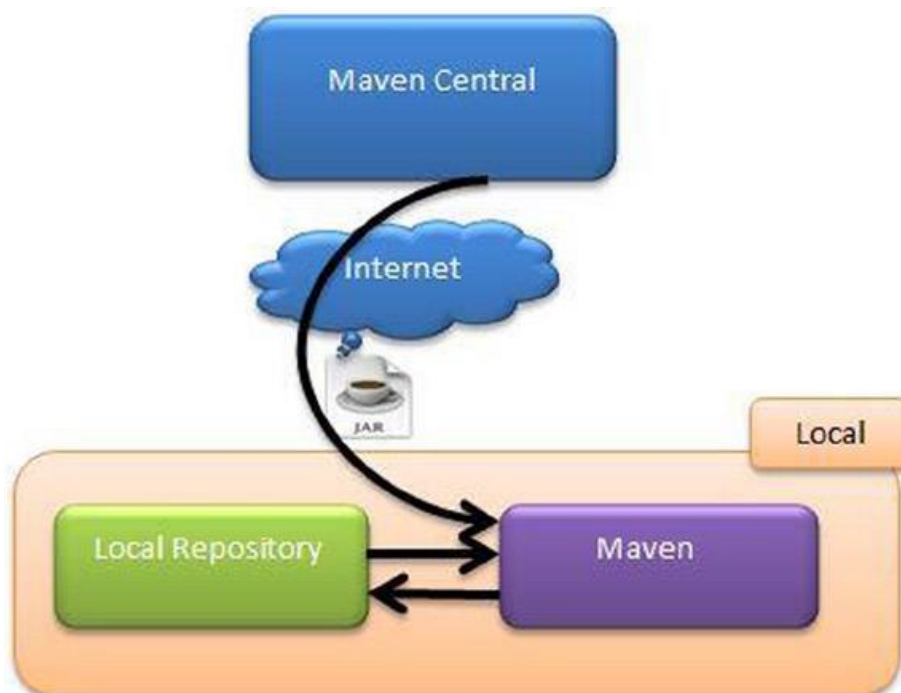
Глава 3: Управление зависимостями Maven

Проекты корпоративного уровня обычно зависят от множества библиотек с открытым исходным кодом. Представьте ситуацию, что вы захотели использовать **Log4J** для журналирования своего приложения. Чтобы добиться этого вы бы зашли на страницу скачивания **Log4J**, скачали бы **JAR-файл** и разместили бы его в папке **lib** своего проекта или добавили бы его в путь к классам. Но как вы уже знаете, при таком подходе существует пара проблем:

1. Вы должны зарегистрировать **JAR-файл** в **SVN**, чтобы ваш проект можно было собрать на другом компьютере;
2. **JAR-файл**, который вы скачали, может зависеть от ряда других библиотек. Тогда вы должны отыскать все эти зависимости и тоже добавить их в свой проект;
3. Когда придет время обновить **JAR-файл**, вы должны будете повторить весь процесс снова;
4. Все это затрудняет обмен **JAR-файлами** между командами вашей организации.

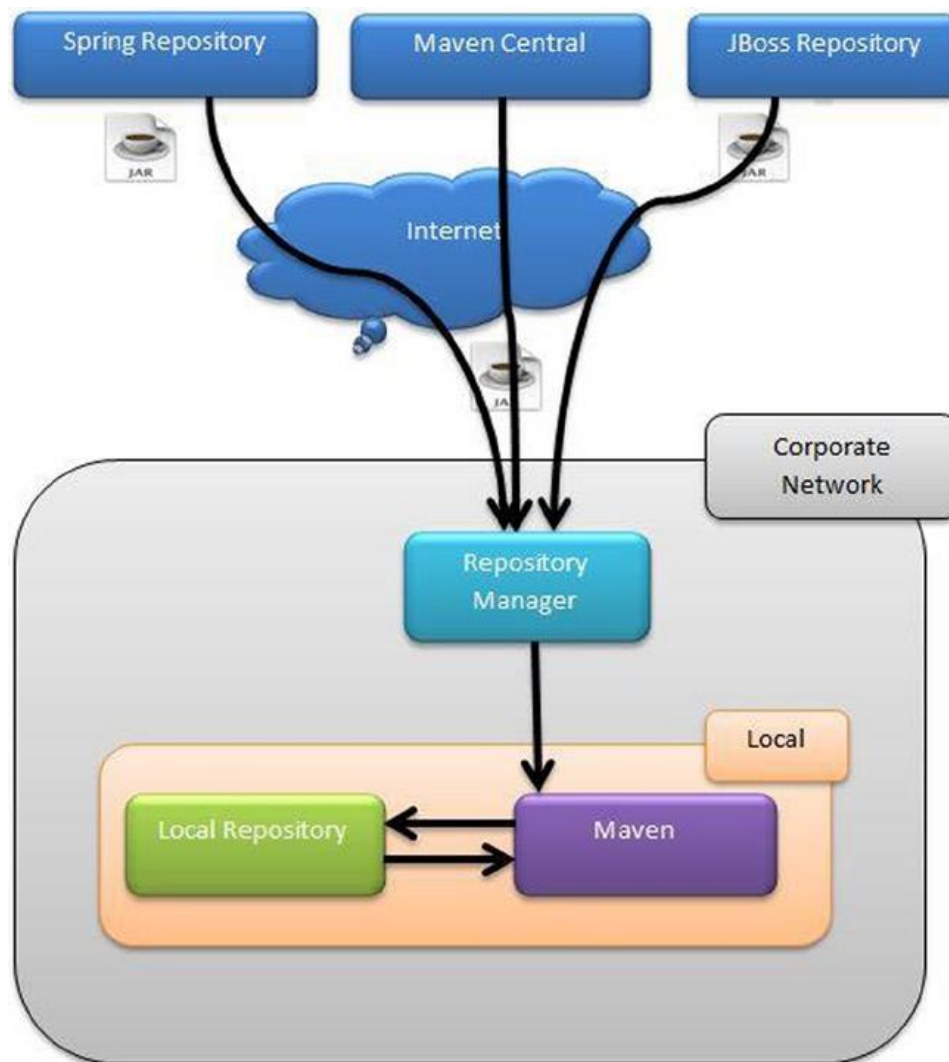
Для решения этих проблем **Maven** обеспечивает декларативное управление зависимостями. При таком подходе вы объявляете зависимости вашего проекта во внешнем файле с именем **pom.xml**. **Maven** автоматически скачает эти зависимости и привяжет их к вашему проекту для последующей сборки, тестирования или упаковки.

Рис. 3-1 отображает общий взгляд на управление зависимостями **Maven**. Как вы видите, **Maven** взаимодействует с хранилищами, содержащими сущности и связанные с ними метаданные. Хранилища, которые обычно доступны через сеть, рассматриваются как удаленные и поддерживаются третьей стороной. Удаленное хранилище, по умолчанию используемое Maven, называется **Централом Maven (Maven Central)** и расположено оно по адресам repo.maven.apache.org и uk.maven.org (сейчас **Maven Central** переехал на адрес search.maven.org - прим.перев.). Скачанные сущности (далее - артефакты) Maven размещает локальном хранилище.

Рисунок 3-1. Управление зависимостями **Maven**

Хотя архитектура, изображенная на **Рис.3-1**, в большинстве случаев работает, но создает несколько проблем в корпоративной среде. Первая проблема заключается в том, что невозможно совместное использование артефактов компании между командами этой же компании. По соображениям безопасности и интеллектуальной собственности вам бы не захотелось опубликовывать сущности вашей компании на **Централе Maven**. Следующая проблема касается проблем законности и лицензирования. Ваша компания может требовать, чтобы команды использовали только официально утвержденное открытое программное обеспечение, а эта архитектура не вписывается в подобную модель. Последняя проблема касается пропускной способности и скорости скачивания. В периоды большой нагрузки на **Централ Maven** скорость скачивания сущностей **Maven** ограничена, что может оказать негативное влияние на ваши сборки. Поэтому большинство предприятий используют архитектуру, изображенную на **Рис.3-2**.

Рис. 3-2. Архитектура хранилища **Maven** для предприятия



Внутренний менеджер хранилища выступает в качестве прокси к удаленным хранилищам. По причине того, что вы полностью контролируете внутреннее хранилище, вы можете управлять типом сущностей, допустимых в вашей компании. Кроме того, вы можете разместить сущности вашей организации на сервере, тем самым разрешая совместную работу с ними. Существует несколько менеджеров хранилищ с открытым исходным кодом. В **Таблице 3-1** перечислены некоторые из них.

Таблица 3-1. Менеджеры хранилищ с открытым исходным кодом

Менеджер хранилища	URL
Sonatype Nexus	www.sonatype.com/nexus
Apache Archiva	http://archiva.apache.org/
Artifactory	www.jfrog.com/open-source/

Использование хранилищ

Для того чтобы использовать новое хранилище вам нужно изменить файл [settings.xml](#). **Листинг 3-2** отображает хранилища **Spring** и **JBoss**, добавленные в файл [settings.xml](#). Таким же образом вы можете добавить менеджер хранилища вашей компании.

Листинг 3-1. Добавление Хранилищ в settings.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<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">
  <profiles>
    <profile>
      <id>your_company</id>
      <repositories>
        <repository>
          <id>spring_repo</id>
          <url>http://repo.spring.io/release/</url>
        </repository>
        <repository>
          <id>jboss_repo</id>
          <url>https://repository.jboss.org/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>your_company</activeProfile>
  </activeProfiles>
</settings>
```

Замечание: Информация о хранилищах может быть предоставлена как в [settings.xml](#), так и в [pom.xml](#) файле. Существуют аргументы как за, так и против каждого из подходов. Размещение информации о хранилище в файле [pom.xml](#) может сделать сборки более компактными. Такой подход позволяет разработчикам скачивать проекты и просто собирать их без всякой последующей модификации их локальных [settings.xml](#) файлов. Проблема такого подхода в том, что, когда сущности опубликованы, соответствующие файлы [pom.xml](#) будут содержать в себе жестко прописанную информацию о хранилище. Если URL хранилища будет когда-либо изменен, пользователи этих сущностей столкнутся с ошибками по причине того, что пути к хранилищам будут некорректными. Размещение же информации о хранилищах в файле [settings.xml](#) решает эту проблему, и по причине гибкости такого подхода использование [settings.xml](#) обычно и рекомендуется правилами предприятия.

Идентификация зависимости

Зависимости Maven обычно архивируются как **JAR**, **WAR**, архивы предприятия (**EAR**) или **ZIP**. Каждая зависимость **Maven** уникально идентифицируется при помощи следующих координат с использованием группы (**Group**), сущности (**Artifact**) и версии (**Version**) - **GAV**:

groupId: Идентификатор организации или группы, отвечающей за этот проект. Например, **org.hibernate**, **log4j**, **org.springframework.boot**;

artifactId: Идентификатор сущности, сгенерированный проектом. Он должен быть уникальным среди проектов, с таким же **groupId**. Например, **hibernate-tools**, **log4j**, **spring-core** и т.д.;

version: Отображает номер версии проекта. Например, **1.0.0**, **2.3.1-SNAPSHOT** и **4.3.6.Final**.

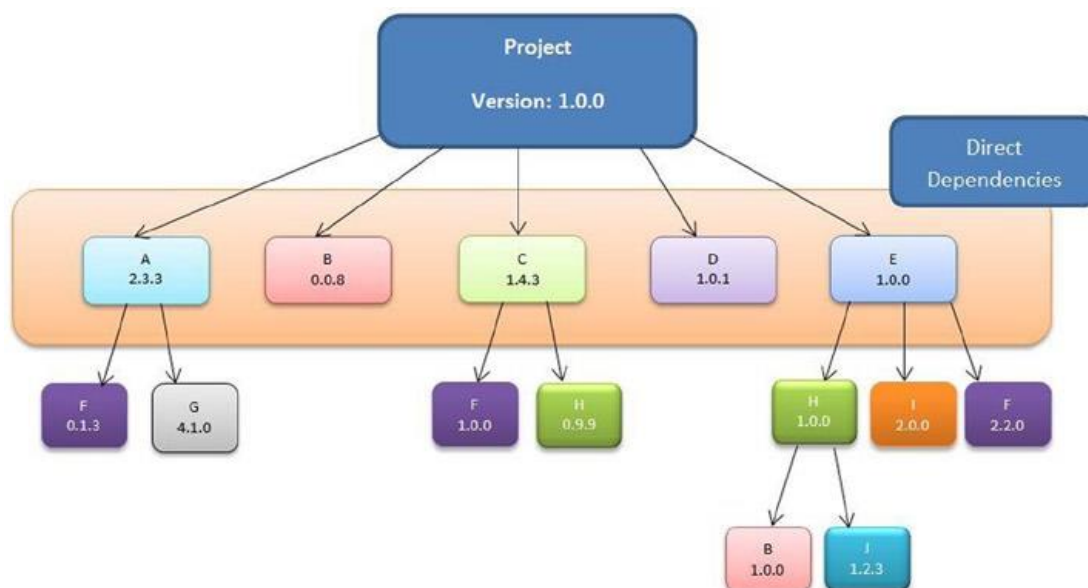
type: Отображает контейнер созданной сущности. Примеры включают в себя **JAR**, **WAR** и **EAR**.

Транзитивные зависимости

Зависимости, объявленные в файле [pom.xml](#) проекта, часто имеют свои собственные зависимости. Такие зависимости называются **транзитивными**. Возьмем, например, **Hibernate Core**. Для правильной работы она требует **JBoss Logging**, **dom4j**, **javaassist** и прочего. **Hibernate Core** объявленная в вашем файле [pom.xml](#) рассматривается как прямая зависимость, а такие зависимости, как **dom4j** и **javaassist** являются транзитивными зависимостями. Важное преимущество **Maven** состоит в том, что он автоматически обрабатывает транзитивные зависимости и включает их в ваш проект.

Рис.3-3 дает пример транзитивных зависимостей. Обратите внимание, что транзитивные зависимости могут иметь свои собственные зависимости. Как вы догадываетесь всё это может быстро стать сложным, особенно когда множественные прямые зависимости ведут к разным версиям одного и того же [JAR-файла](#).

Рисунок 3-3. Транзитивные зависимости



Для разрешения конфликтов версий **Maven** использует подход, известный как **посредничество зависимостей**. Проще говоря, посредничество зависимостей позволяет **Maven** получать зависимость, находящуюся ближе всего в дереве. На **Рис. 3-3** присутствует две версии зависимости «**B**»: «**0.0.8**» и «**1.0.0**». В этом случае версия «**0.0.8**» зависимости «**B**» включается в проект, потому что она является прямой зависимостью и ближайшей в дереве. Теперь взглянем на дерево версий зависимости «**F**»: «**0.1.3**», «**1.0.0**» и «**2.2.0**». Все три зависимости находятся на одинаковом уровне. В этом случае Maven будет использовать **первую найденную** зависимость, которой оказывается «**0.1.3**», а не последняя версия «**2.2.0**».

Замечание: Являясь крайне полезными, тем не менее транзитивные зависимости могут стать причиной проблем и непредсказуемых побочных эффектов, которые могут привести к включению нежелательных **JAR**-файлов или их более старых версий. Всегда анализируйте ваше дерево зависимостей и убеждайтесь, что вы используете именно те зависимости, которые вам нужны, и исключаете те, которые вам не требуются.

Область видимости зависимости

Рассмотрим проект, использующий **JUnit** для тестирования своих модулей. **JAR-файл JUnit**, который вы включаете в свой проект, требуется только в процессе тестирования. И в финальном архиве проекта вам этот **JAR-файл JUnit**-а не нужен. Похожая ситуация с драйвером базы данных **MySQL**, файлом [mysql-connector-java.jar](#). Эта зависимость нужна только при запуске приложения внутри такого контейнера, как **Tomcat**. **Maven** использует концепцию области видимости, позволяющую указать, когда и где вам определенная зависимость требуется.

Maven предоставляет следующие шесть областей видимости:

compile: Зависимости с областью видимости **compile** доступны в пути к классам на всех фазах построения, тестирования и запуска проекта. Это область видимости по умолчанию;

provided: Зависимости с областью видимости **provided** доступны в пути к классам в фазах сборки и тестирования проекта. Они не идут в комплекте с созданными артефактами. Примеры зависимостей с такой областью видимости - **Servlet** api, **JSP** api, и пр.;

runtime: Зависимости с областью видимости **runtime** недоступны в пути к классам в течение фазы сборки. Вместо этого они связываются с созданными артефактами и доступны во время выполнения приложения;

test: Зависимости с областью видимости **test** доступны на фазе тестирования. **JUnit** и **TestNG** являются хорошими примерами зависимостей с тестовой областью видимости;

system: Зависимости с областью видимости **system** подобны зависимостям с областью видимости **provided** за исключением того, что эти зависимости не извлекаются из репозитория. Вместо этого пути, из которых используются эти зависимости, жестко заданы в коде;

import: Область видимости **import** применяется только для зависимостей **.pom-файла**. Позволяет включать информацию управления зависимостями из отдельного **.pom-файла**. Эта область видимости доступна только начиная с **Maven 2.0.9**.

Ручная установка зависимостей

В идеале вы будете внедрять в ваш проект зависимости из публичных хранилищ или от менеджера репозитория предприятия. В любом случае будут случаи, когда для продолжения работы вам понадобится архив, доступный из локального хранилища. Например, вы можете ожидать, что ваши системные администраторы добавят требуемый **JAR-файл** в менеджер репозитория предприятия.

Maven предоставляет удобный способ инсталляции архива в локальное хранилище с помощью соответствующего плагина **install**. **Листинг 3-2** демонстрирует инсталляцию файла **test.jar**, расположенного в папке **c:\apress\gswm-book\chapter3**.

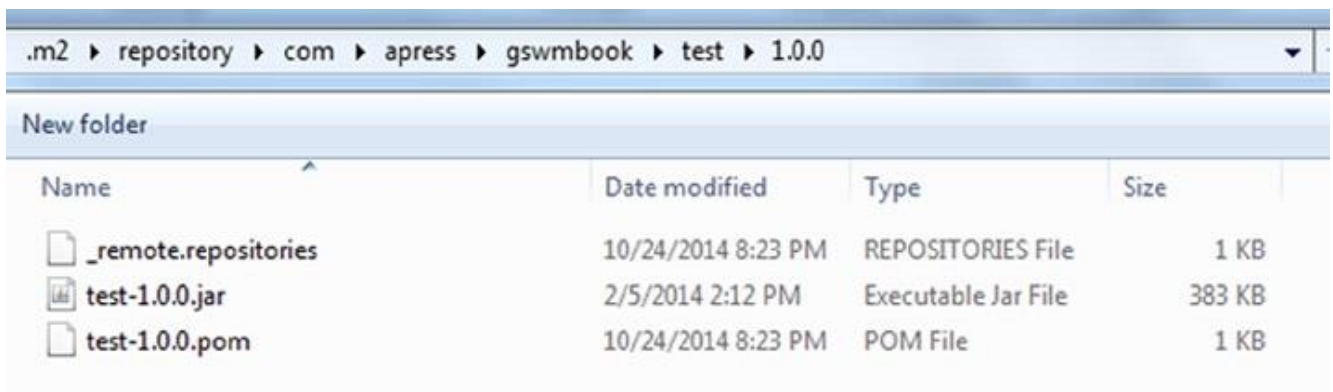
Листинг 3-2. Установка зависимости вручную

```
C:\apress\gswm-book\chapter3>mvn install:install-file -DgroupId=com.apress.gswmbook
-DartifactId=test -Dversion=1.0.0 -Dfile=C:\apress\gswm-book\chapter3\test.jar
-Dpackaging=jar -DgeneratePom=true
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
```

```
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom-
[INFO] Installing C:\apress\gswm-book\chapter3\test.jar to C:\Users\<<USER_
NAME>>\.m2\repository\com\apress\gswmbook\test\1.0.0\test-1.0.0.jar
[INFO] Installing C:\Users\<<USER_NAME>>\AppData\Local\Temp\ mvninstall12668943665146984418.pom
to C:\Users\<<USER_NAME>>\.m2\repository\ com\apress\gswmbook\test\1.0.0\test-1.0.0.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.470 s
[INFO] Finished at: 2014-10-24T20:23:36-06:00
[INFO] Final Memory: 4M/15M
```

После появления сообщения **BUILD SUCCESS** вы можете проверить установку, проследовав в локальное хранилище **Maven**, как показано на **Рис.3-4**.

Рисунок 3-4. Зависимость, добавленная в хранилище



Name	Date modified	Type	Size
_remote.repositories	10/24/2014 8:23 PM	REPOSITORIES File	1 KB
test-1.0.0.jar	2/5/2014 2:12 PM	Executable Jar File	383 KB
test-1.0.0.pom	10/24/2014 8:23 PM	POM File	1 KB

Итоги

Управление зависимостями является сердцем **Maven**. Каждый нетривиальный **Java**-проект полагается на открытые источники или внешние артефакты и управление зависимостями **Maven** автоматизирует процесс получения этих артефактов и включения их на нужных стадиях процесса построения. Также вы знаете, что **Maven** использует **GAV**-координаты для идентификации своих артефактов.

В следующей главе вы узнаете об устройстве базового **Maven**-проекта.

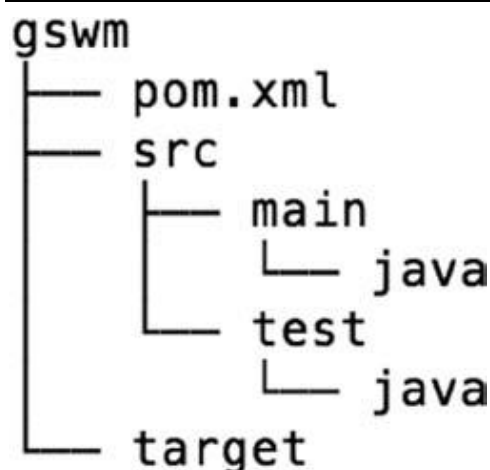
Глава 4: Основы Maven-проекта

Maven следует соглашениям и стандартному расположению папок для всех своих проектов. Как уже обсуждалось в **Главе 1**, такая стандартизация предоставляет единый интерфейс построения, а также облегчает разработчикам переход с одного проекта на другой. В этой главе будут объяснены основы **Maven**-проекта и файл [pom.xml](#).

Организация основы проекта

Лучший способ понять структуру **Maven**-проекта, это взглянуть на неё. На **рис. 4-1** изображен скелет **Java**-проекта, основанного на **Maven**.

Рис. 4-1. Структура **Java**-проекта, основанного на **Maven**



Рассмотрим каждый из компонентов проекта:

- [gswm](#) является корневой папкой проекта. Обычно имя проекта совпадает с именем созданного артефакта;
- папка [src](#) содержит артефакты, связанные с проектом, которыми обычно управляют в системе контроля версий (**SCM**), таких как **SVN** или **Git**;
- папка [src/main/java](#) содержит исходный код **Java**;
- папка [src/test/java](#) содержит код модульных тестов **Java**;
- папка [target](#) содержит созданные артефакты, такие, как [.class-файлы](#). Созданные артефакты обычно не хранятся в **SCM**, поэтому вам не нужно индексировать эту папку и её содержимое в **SCM**;

- каждый **Maven**-проект в корне проекта содержит файл [pom.xml](#). Он содержит такую информацию о проекте и его конфигурации, как его зависимости и плагины.

В дополнение к папкам [src/main](#) и [src/test](#) **Maven** предлагает и несколько других директорий. В **таблице 4-1** перечислены эти директории вместе с их содержимым.

Таблица 4-1. Директории **Maven**

Имя директории	Описание
src/main/resources	Содержит такие ресурсы, как конфигурационные файлы Spring и шаблоны Velocity , которые должны оказаться в созданном артефакте.
src/main/config	Содержит конфигурационные файлы, такие как контекстные файлы Tomcat , конфигурационные файлы James Server и т.д. Эти файлы не попадают в сгенерированный артефакт.
src/main/scripts	Содержит любые скрипты, требующиеся системным администраторам и разработчикам в приложении.
src/test/resources	Содержит конфигурационные файлы , необходимые для тестирования.
src/main/webapp	Содержит веб-ресурсы , такие как .jsp-файлы , таблицы стилей и изображения.
src/it	Содержит интеграционные тесты для приложения.
src/main/db	Содержит файлы базы данных , такие, как SQL-скрипты.
src/site	Содержит файлы, необходимые при генерации сайта проекта.

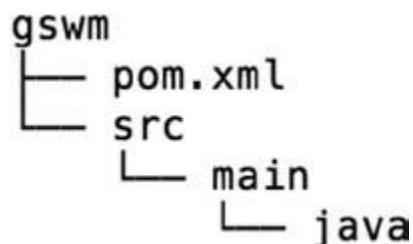
Maven использует идею архетипов (обсуждается в **Главе 6**) для быстрой загрузки проектов. Однако в этой главе мы будем создавать проект вручную. Для создания проекта следуйте следующим инструкциям:

1. Используя командную строку переместитесь в папку, в которой вы хотите создать проект. В данной книге мы подразумеваем, что такой директорией станет [c:\apress\gswm-book\chapter4](#).
2. Выполните команду `mkdir gswm`.
3. Переместитесь командой `cd` в только что созданную директорию и создайте пустой файл [pom.xml](#).

4. Создайте директорию `src/main/java`: создайте директорию `src` в каталоге `gswm` и потом создайте директорию `main` в директории `src` и, в конце концов, создайте директорию `java` в папке `main`, как показано на **Рис.4-2**.

Структура первоначального проекта должна быть подобна изображенной на **Рис.4-2**.

Рис. 4-2. Структура проекта Maven



Содержимое файла `pom.xml`

Файл `pom.xml` является единственным необходимым в Maven-проекте артефактом. Как уже обсуждалось в этой книге файл `pom.xml` содержит конфигурационную информацию, необходимую для Maven. **Листинг 4-1** отображает файл `pom.xml` с базовой информацией о проекте. Файл `pom.xml` начинается с элемента `project`. Потом указываются координаты `groupId`, `artifactId` и `version`. Элемент `packaging` сообщает **Maven** что ему нужно создавать JAR-архив для проекта. В завершение добавляется информация о разработчиках, работающих над проектом.

Листинг 4-1. Конфигурация файла `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.apress.gswmbook</groupId>
  <artifactId>gswm</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Getting Started with Maven</name>
  <url>http://apress.com</url>
  <developers>
    <developer>
      <id>balaji</id>
      <name>Balaji Varanasi</name>
      <email>balaji@inflinx.com</email>
      <properties>
        <active>true</active>
      
```

```

    </properties>
  </developer>
  <developer>
    <id>sudha</id>
    <name>Sudha Belida</name>
    <email> sudha@inflinx.com</email>
    <properties>
      <active>true</active>
    </properties>
  </developer>
</developers>
</project>

```

Мы будем рассматривать другие элементы файла `pom.xml` в этой главе позже, а также на протяжении всей оставшейся книги.

ОБОЗНАЧЕНИЕ ВЕРСИЙ В MAVEN

В проекте Maven рекомендуется использовать следующее соглашение для обозначения версий:

```
<major-version>.<minor-version>.<incremental-version>-qualifier
```

Старшее (major), **младшее (minor)** и **дополняющее (incremental)** значения являются числами, а **квалификатор (qualifier)** принимает такие значения, как **RC**, **alpha**, **beta** и **SNAPSHOT**. Примеры нумерации, следующей этому соглашению: **1.0.0**, **2.4.5-SNAPSHOT**, **3.1.1-RC1** и т.д.

Квалификатор **SNAPSHOT** в версии проекта имеет специальное значение. Он показывает, что проект находится в стадии разработки. Когда проект использует зависимость **SNAPSHOT** то при каждом построении проекта **Maven** будет получать и использовать самый последний из артефактов **SNAPSHOT**.

Большинство менеджеров хранилищ принимают сборку релиза лишь единожды. Однако, если вы разрабатываете приложение в среде с непрерывной интеграцией, то сборки вам нужно производить часто и в менеджере хранилища размещать последнюю из них. Следовательно, суффикс **SNAPSHOT** является наилучшим способом пометить версию, находящуюся в процессе разработки.

Сборка проекта

Перед рассмотрением сборки проекта давайте добавим **Java**-класс *HelloWorld* в папку `src/main/java`. В **Листинге 4-2** приведен код для класса *HelloWorld*.

Листинг 4-2. Код Java-класса *HelloWorld*

```
public class HelloWorld {
    public void sayHello() {
        System.out.print("Hello World");
    }
}
```

Рисунок 4-3 показывает структуру проекта после добавления класса *HelloWorld*.

Рис. 4-3. Структура проекта с добавленным Java-классом

```
gswm
├── pom.xml
├── src
│   └── main
│       └── java
│           └── HelloWorld.java
```

Теперь вы готовы построить приложение поэтому запустите `mvn package` из папки `gswm`. Вы должны увидеть вывод, похожий на тот, что изображен в **Листинге 4-3**.

Листинг 4-3. Вывод команды **Maven Package** при сборке проекта

```
C:\apress\gswm-book\chapter4\gswm>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----mvn -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources)
@ gswm
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,i.e. build is
platform dependent! [INFO] skip non existing resourceDirectory C:\apress\gswm-book\chapter4\
gswm\src\main\resources [INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile)
```

```

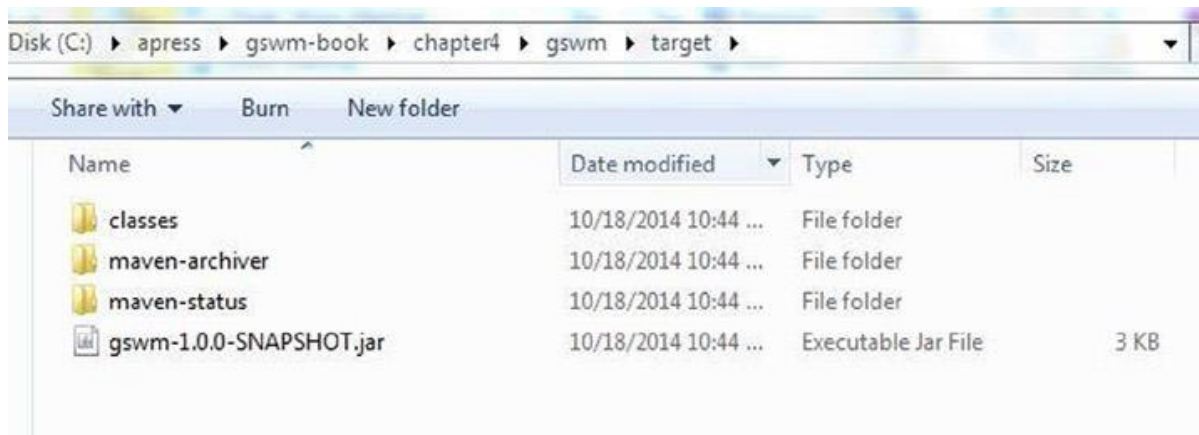
@ gswm -
[WARNING] File encoding has not been set, using platform encoding Cp1252,
i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\apress\gswm-book\chapter4\gswm\target\ classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources)
@ gswm -
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,i.e. build is
platform dependent!
[INFO] skip non existing resourceDirectory C:\apress\gswm-book\chapter4\
gswm\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile)
@ gswm-
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test)
@ gswm -
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar)
@ gswm -
[INFO] Building jar: C:\apress\gswm-book\chapter4\gswm\target\gswm-1.0.0SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.269s
[INFO] Finished at: Mon Oct 13 21:40:44 MDT 2014
[INFO] Final Memory: 9M/23M
[INFO] -----

```

Замечание: Если вы запускаете **Maven** впервые, то он скачает плагины и зависимости, необходимые для своего запуска, поэтому ваша первая сборка может занять больше времени, чем вы ожидаете.

Суффикс **package** после команды **mvn** это фаза **Maven** на которой **Java**-код компилируется и упаковывается в **JAR-файл**. Упакованный **JAR-файл** создается в папке **gswm\target**, как это показано на **Рисунке 4-4**.

Figure 4-4. Упакованный **JAR**-файл, расположенный в папке **target**



Тестирование проекта

Теперь, когда вы завершили сборку проекта, давайте добавим **JUnit**-тест, который протестирует метод `sayHello()`. Начнем эту процедуру с добавления зависимости **JUnit** в файл `pom.xml`. Достичь этого вы можете с использованием элемента **dependencies**. **Листинг 4-4** показывает обновленный `pom.xml` файл с зависимостью **JUnit**.

Listing 4-4. Обновленный файл `pom.xml` с зависимостью **JUnit**

```
<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.apress.gswmbook</groupId>
  <artifactId>gswm</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Getting Started with Maven</name>
  <url>http://apress.com</url>
  <developers>
    <developer>
      <id>balaji</id>
      <name>Balaji Varanasi</name>
      <email>balaji@inflinx.com</email>
      <properties>
        <active>>true</active>
      </properties>
    </developer>
    <developer>
      <id>sudha</id>
```

```

    <name>Sudha Belida</name>
    <email>sudha@inflinx.com</email>
    <properties>
      <active>true</active>
    </properties>
  </developer>
</developers>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>

```

Обратите внимание, что мы применили параметр области видимости **test**, указывающий, что **JUnit.jar** нужен только в течение фазы тестирования. Давайте убедимся, что эта зависимость была успешно добавлена путем вызова команды **mvn dependency:tree** в командной строке. **Листинг 4-5** показывает результат выполнения этой операции.

Листинг 4-5. Результат вызова команды **mvn dependency:tree**

```

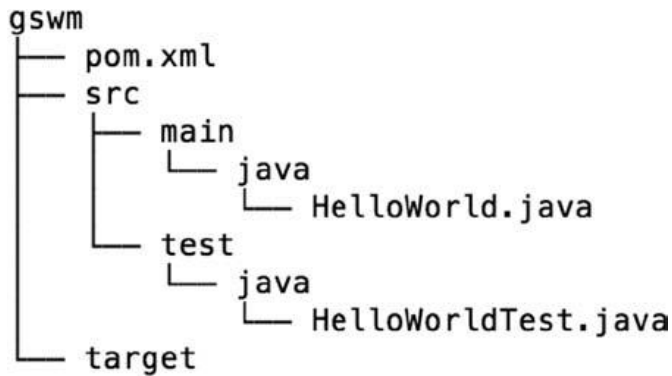
C:\apress\gswm-book\chapter4\gswm>mvn dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-dependency-plugin:2.8:tree (default-cli) @ gswm ---
[INFO] com.apress.gswmbook:gswm:jar:1.0.0-SNAPSHOT
[INFO] \- junit:junit:jar:4.11:test
[INFO] \- org.hamcrest:hamcrest-core:jar:1.3:test
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.646s
[INFO] Finished at: Mon Oct 13 21:54:24 MDT 2014
[INFO] Final Memory: 7M/19M
[INFO] -----

```

Цель **tree** в плагине зависимостей отображает зависимости проекта в виде дерева. Обратите внимание, что зависимость **JUnit** извлекла транзитивную зависимость с названием **hamcrest**, которая является проектом с открытым исходным кодом, облегчающим запись совпадающих объектов.

Теперь, когда у нас есть зависимость **JUnit** в пути к классам, давайте добавим в проект модульный тест `HelloWorldTest.java`. Создайте папку `test/java` в каталоге `src` и добавьте в него файл `HelloWorldTest.java`. Обновленная структура проекта изображена на **Рисунке 4-5**.

Рисунок 4-5. Структура Maven с тестовым классом



Исходный код для `HelloWorldTest` показан в **Листинге 4-6**.

Листинг 4-6. Код класса HelloWorldTest

```

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class HelloWorldTest {
    private final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

    @Before
    public void setUp() {
        System.setOut(new PrintStream(outputStream));
    }

    @Test
    public void testSayHello() {
        HelloWorld hw = new HelloWorld();
        hw.sayHello();
        Assert.assertEquals("Hello World", outputStream.toString());
    }

    @After
    public void cleanUp() {
        System.setOut(null);
    }
}
  
```

```

    }
}

```

Теперь, когда у вас в проекте всё настроено, можно выполнить команду **mvn package** еще раз. После того, как вы её запустите, вы увидите вывод, похожий на изображенный в **Листинге 4-7**.

Листинг 4-7. Вывод команд Maven при сборке проекта

```

C:\apress\gswm-book\chapter4\gswm>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources mvn(default-resources)
@ gswm --
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,i.e. build is
platform dependent!
[INFO] skip non existing resourceDirectory C:\apress\gswm-book\chapter4\
gswm\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ gswm --
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources)
@ gswm --
-----
[INFO] Surefire report directory: C:\apress\gswm-book\chapter4\gswm\target\ surefire-reports
-----
T E S T S
-----
Running HelloWorldTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.038 sec Results : Tests run:
1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ gswm --
[INFO] Building jar: C:\apress\gswmbook\chapter4\gswm\target\gswm-1.0.0SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.807s
[INFO] Finished at: Mon Oct 13 21:59:57 MDT 2014
[INFO] Final Memory: 9M/22M
[INFO] -----

```

Обратите внимание на секцию **TESTS** в **Листинге 4-7**. Она показывает, что Maven запустил тест, и он был успешно завершен.

Рисунок 4-6. Папка `target` с классами тестов

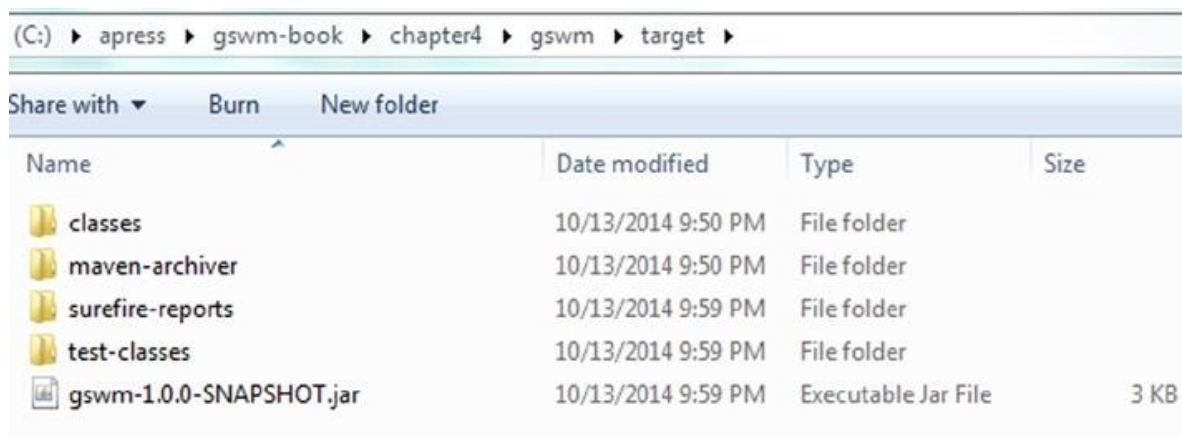


Рисунок 4-6 отображает обновленную папку `target`. Можно заметить, что теперь у вас есть папка `test-classes`, содержащая скомпилированные файлы тестовых классов.

Свойства в `pom.xml`

Maven позволяет вам объявлять свойства в файле `pom.xml` с помощью элемента `<properties>`. Эти свойства весьма полезны для объявления версий зависимостей. **Листинг 4-8** отображает обновленный файл `pom.xml` с версией **JUnit**, объявленной как свойство. Обратите внимание на использование синтаксической конструкции `${}` в элементе версии зависимости **JUnit**. Она особенно полезна, когда `pom.xml` имеет множество зависимостей и вам нужно выяснить или изменить версию какой-либо зависимости.

Листинг 4-8. Файл `pom.xml` со свойствами

```
<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.apress.gswmbook</groupId>
  <artifactId>gswm</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Getting Started with Maven</name>
  <url>http://apress.com</url>
  <properties>
```

```

    <junit.version>4.11</junit.version>
  </properties>
  <developers>
    <developer>
      <id>balaji</id>
      <name>Balaji Varanasi</name>
      <email>balaji@inflinx.com</email>
      <properties>
        <active>true</active>
      </properties>
    </developer>
    <developer>
      <id>sudha</id>
      <name>Sudha Belida</name>
      <email>sudha@inflinx.com</email>
      <properties>
        <active>true</active>
      </properties>
    </developer>
  </developers>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

Исключение зависимостей

В **Главе 3** обсуждались транзитивные зависимости и частная необходимость исключить конкретную транзитивную зависимость. Элемент **exclusions** в файле [pom.xml](#) позволяет исключить некоторую зависимость.

Листинг 4-9 отображает обновленный элемент зависимостей **JUnit**, где транзитивная зависимость **hamcrest** исключена. Как вы видите, элемент **exclusion** принимает координаты **groupId** и **artifactId** той зависимости, которую вам нужно исключить.

Листинг 4-9. Зависимость JUnit с исключением

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

ИТОГИ

Соглашения **Maven** предписывают стандартный набор директорий для всех своих проектов. Эти соглашения предусматривают несколько ключевых директорий, таких как `src/main/java` и `src/test`, а также рекомендации относительно их содержимого. Вы узнали об обязательном файле `pom.xml` и о некоторых его элементах, используемых для конфигурирования проекта **Maven**.

В следующей главе мы рассмотрим жизненный цикл **Maven**, плагины, фазы сборки, цели и вопросы их эффективного использования.

Глава 5: Жизненный цикл Maven

Цели и плагины

Процессы построения, генерирующие артефакты, обычно требуют нескольких шагов и заданий для успешного завершения. Примеры таких задач включают компилирование исходного кода, запуск модульного теста и упаковка артефактов. **Maven** использует концепцию целей для представления таких гранулированных задач.

Для лучшего понимания того, что представляют собой цели, рассмотрим пример. **Листинг 5-1** отображает цель **compile**, выполненную над кодом проекта **gswm** в каталоге `C:\apress\gswm-book\chapter5\gswm`. Как и подразумевает само название, цель **compile** компилирует исходный код. Цель **compile** обнаруживает в папке `src\main\java` Java-класс `HelloWorld.java`, компилирует его и размещает скомпилированный класс в папке `target\classes`.

Listing 5-1. Цель Maven compile

```
C:\apress\gswm-book\chapter5\gswm>mvn compiler:compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-cli) @ gswm ---
[WARNING] File encoding has not been set, using platform encoding Cp1252,
i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\apress\gswm-book\chapter5\gswm\target\ classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.197s
[INFO] Finished at: Mon Oct 13 22:11:42 MDT 2014 [INFO] Final Memory: 7M/18M
[INFO] -----
```

Цели в **Maven** упакованы в **плагины**, которые по сути являются коллекциями из одной или более целей. В **Листинге 5-1** **maven-compiler-plugin** - это плагин который предоставляет цель **compile**. **Листинг 5-2** демонстрирует цель под названием **clean**. Как уже упоминалось ранее, директория `target` содержит сгенерированные **Maven** временные файлы и артефакты. Случается, что директория `target` становится слишком большой или из неё нужно удалить некоторые кэшированные файлы. Цель **clean** выполняет именно это, пытаясь удалить папку `target` вместе со всем её содержимым.

Листинг 5-2. Цель Maven clean

```
C:\apress\gswm-book\chapter5\gswm>mvn clean:clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-cli) @ gswm ---
[INFO] Deleting C:\apress\gswm-book\chapter5\gswm\target
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.233s
[INFO] Finished at: Mon Oct 13 22:14:49 MDT 2014
[INFO] Final Memory: 3M/15M
[INFO] -----
```

Обратите внимание на суффикс команды `clean:clean` в Листинге 5-2. Слово `clean` перед двоеточием обозначает плагин `clean`, а `clean` после двоеточия обозначает цель `clean`. Теперь становится очевидным, почему запуск цели из командной строки требует такого синтаксиса:

```
mvn plugin_identifier:goal_identifier
```

Плагины и их поведение могут быть сконфигурированы с использованием секции `<plug-in>` файла `pom.xml`. Рассмотрим случай, когда вам нужно заставить проект компилироваться с помощью **Java 1.6**. Но в версии **3.0** плагин `maven-compiler-plugin` компилирует код с использованием **Java 1.5**. Поэтому вам необходимо модифицировать поведение этого плагина в файле `pom.xml`, как показано в Листинге 5-3.

Листинг 5-3. Элемент `plug-in` в файле `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>
  <!-- Детали проекта опущены ради краткости -->
  <dependencies>
    <!-- Детали зависимостей опущены ради краткости -->
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```

    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
    <configuration>
      <source>1.6</source>
      <target>1.6</target>
    </configuration>
  </plugin>
</plugins>
</build>
</project>

```

Теперь если вы выполните команду **mvn compiler:compile**, то сгенерированные файлы класса будут версии **Java 1.6**.

Замечание: Элемент **<build>** в файле [pom.xml](#) обладает очень полезным дочерним элементом по имени **<finalName>**. По умолчанию имя сгенерированного **Maven** артефакта соответствует формату **<<идентификатор_артефакта_проекта>>-<<версия_проекта>>**. Однако иногда может потребоваться изменить имя сгенерированного артефакта без изменения самого идентификатора **artifactId**. Этого можно добиться путем объявления элемента **<finalName>** как **<finalName>новое_имя</finalName>**.

Жизненный цикл и фазы

Maven следует четкому **жизненному циклу сборки**, когда собирает, тестирует и выдает артефакт. Жизненный цикл образует набор стадий или шагов, которые выполняются в определенном порядке, независимо от создаваемого артефакта. В **Maven** этапы такого жизненного цикла называются **фазами**. **Maven** имеет три встроенных жизненных цикла:

Default: Жизненный цикл поддерживает компиляцию, упаковку и разворачивание проекта **Maven**.

Clean: Этот жизненный цикл осуществляет удаление директории **target**, временных файлов и сгенерированных артефактов.

Site: Этот жизненный цикл поддерживает создание документации и сайта.

Внимание! Сейчас, когда вы уже знаете о жизненном цикле **clean**, вы можете очищать директорию **target** простым запуском фазы очистки, используя команду **mvn clean**.

Для лучшего понимания жизненного цикла сборки и его фаз, давайте рассмотрим некоторые фазы, связанные с жизненным циклом, используемым по умолчанию:

validate: Запускает проверки чтобы убедиться, что проект корректен и что все зависимости загружены и доступны.

compile: Компилирует исходный код.

test: Запускает модульные («юнит») тесты используя фреймворк. Этот шаг не является необходимым для того, чтобы приложение было упаковано.

package: Собирает откомпилированный код в такой формат для распространения, как **JAR** или **WAR**.

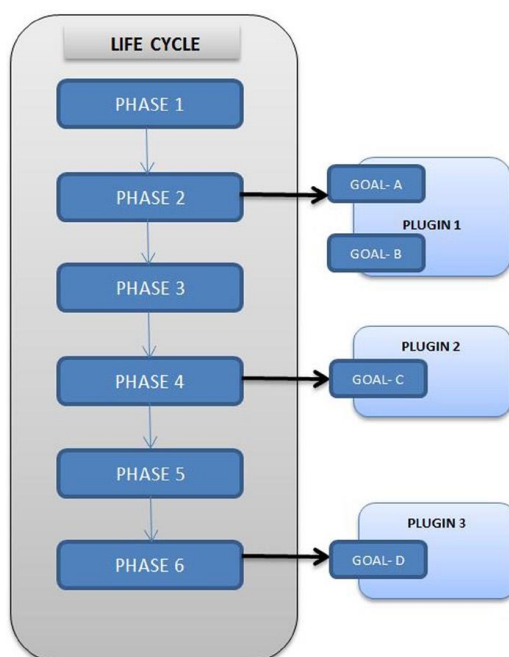
install: Устанавливает упакованный архив в локальный репозиторий. С этого момента архив становится доступным для использования любым проектом, запущенным на данной машине.

deploy: Отправляет собранный архив в удалённое хранилище для использования другими пользователями.

Так как жизненный цикл **default** однозначно определяет порядок фаз, то можно создавать артефакт простым запуском команды **mvn package**. **Maven** автоматически выполнит все предшествующие фазы. В приведенном примере **Maven** запустит фазы **validate**, **compile** и **test** до запуска фазы **package**. Это означает, что разработчикам и менеджерам конфигураций нужно будет выучить и использовать лишь несколько команд.

Многие задачи нужно выполнять на каждой фазе. Для этого каждая фаза ассоциируется с одной или более целями. Фаза просто передает свои задачи связанным с ней целям. **Рисунок 5-1** отображает связи между жизненным циклом, фазами, целями и плагинами.

Рисунок 5-1. Связь между жизненным циклом, фазами, целями и плагинами



Элемент `<packaging>` в файле `pom.xml` будет автоматически назначать нужные цели для каждой из фаз без какого-либо дополнительного конфигурирования. Следует помнить, что в этом и состоит преимущество *Соглашения по конфигурации (CoC)*. Если элемент `<packaging>` содержит необходимый **JAR**, то фаза `package` будет связана с **JAR**-целью в **JAR**-плагине. Подобным же образом, для **WAR**-артефакта файл `pom.xml` привяжет пакет к **WAR**-цели в **WAR**-плагине.

Пропуск тестов

Как уже обсуждалось ранее, при запуске стадии `package` фаза `test` также запускается и исполняются все модульные тесты. Если на тестовой фазе возникают какие-либо проблемы, то сборка прерывается. Это желательное поведение, но бывает, что, например, в унаследованном проекте для сборки проекта вам нужно пропустить запуск тестов. Добиться этого можно используя свойство `maven.test.skip`. Вот пример использования этого свойства:

```
mvn package -Dmaven.test.skip=true
```

Разработка плагинов

Разработка плагинов для **Maven** весьма проста. В этом разделе мы покажем, как разработать пример *HelloPlugin*, чтобы дать вам почувствовать вкус к разработке плагинов.

Как уже обсуждалось ранее, плагин - это просто набор целей. Поэтому, когда мы говорим о разработке плагинов, то, по сути, мы говорим о разработке целей. В **Java** эти цели реализуются с использованием объектов **MOJO**, что расшифровывается как **Maven Old Java Object**, что подобно **Java Plain Old Java Object (POJO)**.

Давайте начнем разработку плагина с создания **Java**-проекта **Maven** под именем `gswm-plugin`, как изображено на **Рисунке 5-2**. Создадим этот проект в первоначальной папке проекта `gswm-plugin`, находящейся в каталоге `c:\apress\gswm-book\chapter5`.

Рисунок 5-2. Проект **Maven** для разработки плагина

```

gswm-plugin
├── pom.xml
├── src
│   └── main
│       └── java
│           └── com
│               └── apress
│                   └── plugins
│                       └── HelloMojo.java

```

Замечание: в этой главе мы создаем проект плагина вручную. **Maven** предоставляет **maven-archetype-mojo**, который даст вам толчок в разработке плагинов. Мы узнаем об архетипах **Maven** в **Главе 6**.

Содержимое файла **pom.xml** приведено в **Листинге 5-4**. Обратите внимание, что типом упаковки является **maven-plugin**. Мы добавили зависимость **maven-plugin-api**, поскольку она нужна для разработки плагинов.

Листинг 5-4. Файл **pom.xml** с зависимостью **maven-plugin-api**

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.apress.plugins</groupId>
  <artifactId>gswm-plugin</artifactId>
  <version>1.0.0</version>
  <packaging>maven-plugin</packaging>
  <name>Simple Hello Plugin</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-plugin-api</artifactId>
      <version>3.2.3</version>
    </dependency>
  </dependencies>
</project>
```

Следующим шагом в процессе разработки является создание **МОЖО**. **Листинг 5-5** отображает код для **HelloMojo**. Как вы видите, реализация очень проста. Мы используем экземпляр **Log** для вывода лога в консоль. В действительности, самая важная часть этого кода содержится внутри секции **Java**-комментариев: `@goal hello`. Используя **Javadoc**-тег `@goal` мы объявляем имя цели как **hello**. Также для предоставления метаданных можно использовать такие аннотации **Java 5** как `@Mojo`. В любом случае это требует изменений в файле **pom.xml**, описанных на веб-сайте **Apache Maven** (<http://maven.apache.org/plugin-tools/maven-plugin-plugin/examples/using-annotations.html>).

Листинг 5-5. Java-класс **HelloMojo**

```
package com.apress.plugins;
import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
import org.apache.maven.plugin.MojoFailureException;
/**
 *
 * @goal hello
```

```

*/
public class HelloMojo extends AbstractMojo{
    public void execute() throws MojoExecutionException, MojoFailureException {
        getLog().info("Hello Maven Plugin");
    }
}

```

Последним шагом в этом процессе является инсталляция плагина в **Maven**-репозиторий. Выполните команду **mvn install** в корне директории и у вас должен получиться вывод, отображенный в **Листинге 5-6**.

Листинг 5-6. Команда Maven install

```

C:\apress\gswm-book\chapter5\gswm-plugin>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Simple Hello Plugin 1.0.0
[INFO] -----
[INFO]
[INFO] --- maven-plugin-plugin:3.2:descriptor (default-descriptor) @ gswm-plugin ---
[INFO] Applying mojo extractor for language: java-annotations
[INFO] Mojo extractor for language: java-annotations found 0 mojo descriptors.
[INFO] Applying mojo extractor for language: java
[INFO] Mojo extractor for language: java found 1 mojo descriptors.
[INFO] Applying mojo extractor for language: bsh
[INFO] Mojo extractor for language: bsh found 0 mojo descriptors.
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ gswm-plugin ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\apress\gswm-book\chapter5\ gswm-
plugin\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ gswm-plugin ---
[INFO] Building jar: C:\apress\gswm-book\chapter5\gswm-plugin\target\gswmplugin-1.0.0.jar
[INFO]
[INFO] --- maven-plugin-plugin:3.2:addPluginArtifactMetadata
(defaultaddPluginArtifactMetadata) @ gswm-plugin ---
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ gswm-plugin ---
[INFO] Installing C:\apress\gswm-book\chapter5\gswm-plugin\target\gswmplugin-1.0.0.jar to
C:\Users\<<USER_NAME>>.m2\repository\com\apress\ plugins\gswm-plugin\1.0.0\gswm-plugin-
1.0.0.jar

```

```
[INFO] Installing C:\apress\gswm-book\chapter5\gswm-plugin\pom.xml to C:\
Users\<<USER_NAME>>\.m2\repository\com\apress\plugins\gswm-plugin\1.0.0\ gswm-plugin-1.0.0.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.788s [INFO] Finished at: Mon Oct 13 22:29:55 MDT 2014
[INFO] Final Memory: 13M/32M
[INFO] -----
```

Теперь можно начать использование этого плагина. Запомните, что синтаксис для запуска любой цели - это **mvn идентификаторПлагина:идентификаторЦели**. Листинг 5-7 показывает данный плагин в работе. Обратите внимание на вывод в консоль текста *Hello Maven Plugin*.

Листинг 5-7. Работа плагина Hello-Plug-in

```
C:\apress\gswm-book\chapter5\gswm-plugin>mvn com.apress.plugins:gswm-plugin:hello
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Simple Hello Plugin 1.0.0
[INFO] -----
[INFO] --- gswm-plugin:1.0.0:hello (default-cli) @ gswm-plugin ---
[INFO] Hello Maven Plugin
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.583s
[INFO] Finished at: Mon Oct 13 22:32:55 MDT 2014
[INFO] Final Memory: 4M/15M
[INFO] -----
```

Итоги

Maven использует архитектуру, основанную на плагинах, позволяющую легко расширять его функциональность. Каждый плагин является набором одной или нескольких целей, которые могут использоваться для выполнения таких задач, как компиляция исходного кода или запуск тестов. **Maven** привязывает цели к фазам. Фазы обычно выполняются в некоторой последовательности как часть жизненного цикла сборки. Также вы ознакомились с основами создания плагинов.

В следующей главе вы познакомитесь с архетипами и узнаете о мультимодульных проектах.

Глава 6: Архетипы Maven

До этого момента вы создавали проект **Maven** вручную, создавая папки и файлы `pom.xml` с нуля. Это может быть утомительным, особенно если вам часто приходится создавать проекты. Для решения этой проблемы **Maven** предоставляет *архетипы*. Архетипы **Maven** - это заготовки проектов, которые позволяют пользователям легко создавать новые проекты.

Архетипы **Maven** также предоставляют удобную базу для обмена опытом и обеспечивают постоянство стандартной структуры директорий **Maven**. Например, предприятие может создать архетип с корпоративной каскадной таблицей стилей (**CSS**), утвержденными библиотеками **JavaScript** и повторно используемыми компонентами. Разработчики, используя эти архетипы для создания проектов, будут автоматически следовать стандартам компании.

Встроенные архетипы

Maven прямо «из коробки» предоставляет разработчикам сотни архетипов. Кроме того, существует множество проектов с открытым исходным кодом предоставляющих дополнительные архетипы, которые можно скачать и использовать. **Maven** также предоставляет архетипы плагинов с целями для создания архетипов и генерации проектов из архетипов.

У плагина архетипов существует цель **generate**, позволяющая просматривать и выбирать необходимый архетип. **Листинг 6-1** отображает результаты запуска цели **generate** из командной строки. Как видите, на выбор предоставлен 491 архетип (*на 2018 год их уже было более 2 тысяч – прим.перев.*). Использование нескольких из них рассматривается в этой главе.

Листинг 6-1. Вызов цели **generate** плагина архетипов **Maven**

```
$mvn archetype:generate
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) @ standalone- pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) @ standalone- pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone- pom
[INFO] Generating project in Interactive mode
```



```
[INFO] No archetype defined. Using maven-archetype-quickstart
(org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
.....
.....
1176: remote -> ru.yandex.qatools.camelot:camelot-plugin (-)
1177: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-archetype (-)
1178: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
1179: remote -> tk.skuro:clojure-maven-archetype (A simple Maven archetype for Clojure)
1180: remote -> tr.com.lucidcode:kite-archetype (A Maven Archetype that allows users to
create a Fresh Kite project)
1181: remote -> uk.ac.rdg.resc.edal-ncwms-based-webapp (-)
1182: local -> com.inflinx.book.ldap:practical-ldap-empty-archetype (-)
1183: local -> com.inflinx.book.ldap:practical-ldap-archetype (-) Choose a number or apply
filter (format: [groupId:]artifactId, case sensitive contains): 491:
```

Создание Веб-проекта

Maven предоставляет архетип **maven-archetype-webapp**, позволяющий сгенерировать веб-приложение. Давайте создадим такое приложение путем вызова следующей команды в папке [C:\apress\gswm-book\chapter6](#):

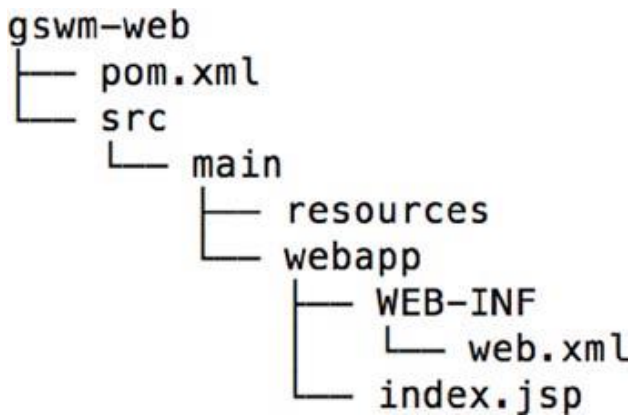
```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp
```

Эта команда выполняется в интерактивном режиме. На поступающие вопросы введите следующую информацию:

```
Define value for property 'groupId': : com.apress.gswmbook
Define value for property 'artifactId': : gswm-web
Define value for property 'version': 1.0-SNAPSHOT: : <<Hit Enter>>
Define value for property 'package': com.apress.gswmbook: : war
Confirm the properties configuration:
groupId: com.apress.gswmbook
artifactId: gswm-web
version: 1.0-SNAPSHOT
package: war
Y: <<Hit Enter>>
```

Сгенерированная структура папок должна походить на ту, что изображена на **Рисунке 6-1**:

Рисунок 6-1. Структура веб-проекта Maven



Файл `pom.xml` минимален и содержит единственную зависимость - **JUnit**. **Maven** может упростить запуск вашего веб-приложения, используя встроенные веб-серверы, такие как **Tomcat** или **Jetty**. Листинг 6-2 отображает модифицированный файл `pom.xml` с добавленным плагином **Tomcat**.

Листинг 6-2. Модифицированный файл `pom.xml` с внедренным плагином **Tomcat**

```

<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.apress.gswmbook</groupId>
  <artifactId>gswm-web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>gswm-web Maven Webapp</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>
  <build>
    <finalName>gswm-web</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.tomcat.maven</groupId>

```

```

        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
    </plugin>
</plugins>
</build>
</project>

```

Для того, чтобы запустить это веб-приложение на сервере **Tomcat** вызовите следующую команду в корневой директории проекта:

```
mvn tomcat7:run
```

Вы увидите развернутый проект и вывод, похожий на изображенный на **Листинге 6-3**.

Листинг 6-3. Вывод команды Tomcat run

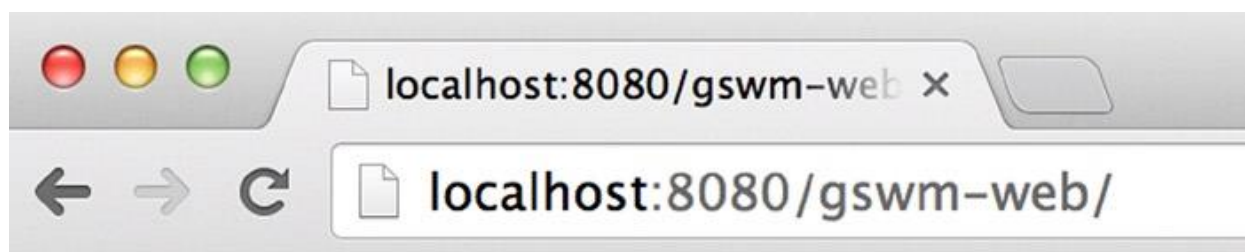
```

Oct 11, 2014 12:08:43 PM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Tomcat
Oct 11, 2014 12:08:43 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.47
Oct 11, 2014 12:08:45 PM org.apache.catalina.util.SessionIdGenerator createSecureRandom
INFO: Creation of SecureRandom instance for session ID generation using
[SHA1PRNG] took [334] milliseconds.
Oct 11, 2014 12:08:45 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]

```

Теперь запустите браузер и перейдите по адресу <http://localhost:8080/gswm-web/>. Вы должны увидеть веб-страницу, подобную изображенной на **Рисунке 6-2**.

Рисунок 6-2. Веб-проект, запущенный в браузере

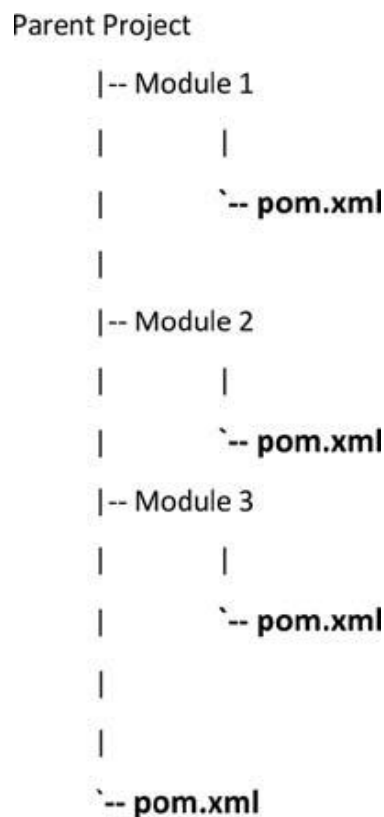


Hello World!

Мультимодульный проект

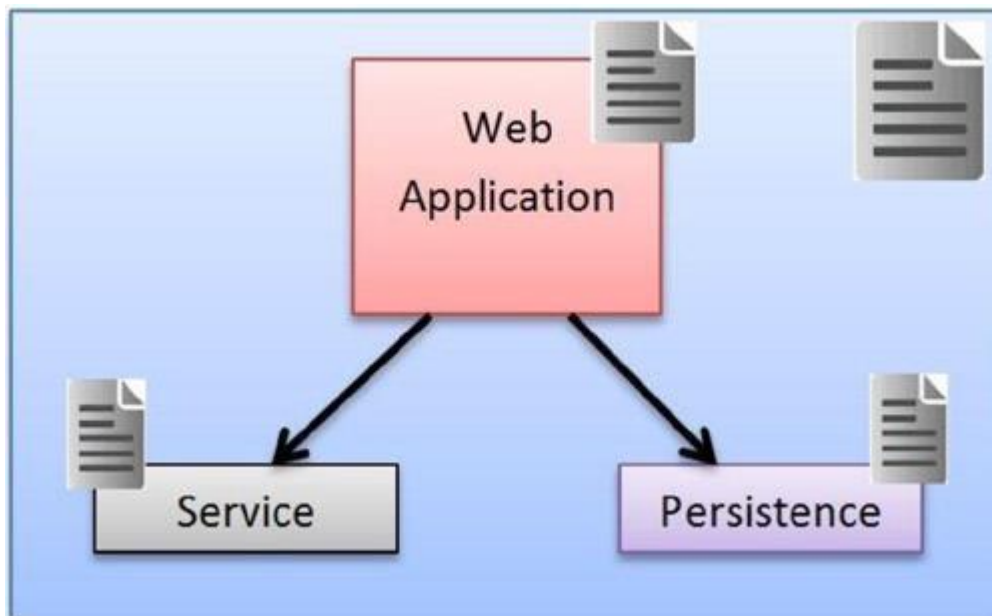
Проекты **Java Enterprise Edition (JEE)** часто разделяются на несколько модулей для облегчения разработки и поддержки. Каждый из этих модулей производит такие артефакты, как **Enterprise JavaBeans (EJBs)**, веб-сервисы, веб-проекты и клиентские **JAR**. **Maven** поддерживает разработку таких больших **JEE**-проектов, позволяя размещать несколько **Maven**-проектов внутри другого **Maven**-проекта. Структура такого мультимодульного проекта отображена на **Рисунке 6-3**. Родительский проект обладает файлом `pom.xml` и несколькими **Maven**-проектами внутри.

Рисунок 6-3. Структура мультимодульного проекта



До конца этой главы мы объясним, как построить мультимодульный проект для задачи, в которой вам нужно разделить большой проект на веб-приложение (**WAR**-артефакт), предоставляющее пользовательский интерфейс, сервисный проект (**JAR**-артефакт), содержащий код сервисного слоя, и проект постоянства (**Persistence**), содержащий код уровня репозитория. На **Рисунке 6-4** отображено визуальное представление такого сценария.

Рисунок 6-4. Мультимодульный проект Maven



Давайте начнём процесс с создания родительского проекта. Чтобы это сделать выполните следующую команду в командной строке в папке `C:\apress\gswm-book\chapter6`:

```

mvn archetype:generate -DgroupId=com.apress.gswmbook -DartifactId=gswm-parent
-Dversion=1.0.0-SNAPSHOT -DarchetypeGroupId=org.codehaus.mojo.archetypes
-DarchetypeArtifactId=pom-root
  
```

Архетип `pom-root` создает папку `gswm-parent` и в ней - файл `pom.xml`. Как следует из Листинга 6-4, созданный файл `pom.xml` имеет минимальное содержимое. Заметьте, что в теге `<packaging>` родительского проекта указан тип `pom`.

Листинг 6-4. Родительский файл `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.apress.gswmbook</groupId>
  <artifactId>gswm-parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>gswm-parent</name>
</project>
  
```

Теперь создайте веб-проект запуском следующей команды в папке [C:\apress\gswm-book\chapter6\gswm-parent](#):

```
mvn archetype:generate -DgroupId=com.apress.gswmbook -DartifactId=gswm-web
-Dversion=1.0.0-SNAPSHOT -Dpackage=war -DarchetypeArtifactId=maven-archetype-webapp
```

В процессе генерации этого веб-проекта вы предоставили Maven такие координаты, как **groupId**, **version** и др. в виде параметров, переданные в генерирующую цель, которая и создала проект **gswm-web**.

Следующим шагом будет создание сервисного проекта. Находясь в папке [C:\apress\gswm-book\chapter6\gswm-parent](#) выполните следующую команду:

```
mvn archetype:generate -DgroupId=com.apress.gswmbook -DartifactId=gswm-service
-Dversion=1.0.0-SNAPSHOT -DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Обратите внимание, что вы не указывали параметр **package**, т.к. **maven-archetype-quickstart** создает **JAR**-проект по умолчанию. Также обратите внимание на использование параметра **interactiveMode**. Он просто указывает Maven выполнять команду, не запрашивая у пользователя никакой информации.

Аналогично предыдущему шагу, создайте следующий **Java**-проект **gswm-repository** путем выполнения в папке [C:\apress\gswm-book\chapter6\gswm-parent](#) следующей команды:

```
mvn archetype:generate -DgroupId=com.apress.gswmbook -DartifactId=gswm-repository
-Dversion=1.0.0-SNAPSHOT -DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Теперь, когда у вас сгенерированы все проекты, взгляните на файл [pom.xml](#) в папке [gswm-parent](#). **Листинг 6-5** отображает его содержимое.

Listing 6-5. Родительский файл [pom.xml](#) с модулями

```
<?xml version="1.0" encoding="UTF-8"?>
<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.apress.gswmbook</groupId>
  <artifactId>gswm-parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>gswm-parent</name>
  <modules>
```

```

    <module>gswm-web</module>
    <module>gswm-service</module>
    <module>gswm-repository</module>
  </modules>
</project>

```

Элемент **<modules>** позволяет объявлять дочерние модули в мультимодульном проекте. По мере генерации каждого модуля **Maven** регистрирует их в качестве дочерних. Кроме того, он модифицирует файлы **pom.xml** самих модулей, добавляя в них информацию о родительском **pom.xml**. **Листинг 6-6** отображает файл **pom.xml** проекта **gswm-web**, в котором указан родительский **pom**-элемент.

Listing 6-6. Файл **pom.xml** веб-модуля

```

<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.apress.gswmbook</groupId>
    <artifactId>gswm-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <groupId>com.apress.gswmbook</groupId>
  <artifactId>gswm-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>gswm-web Maven Webapp</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>
  <build>
    <finalName>gswm-web</finalName>
  </build>
</project>

```

Теперь, когда вся инфраструктура установлена, можно собрать следующий проект. Просто запустите команду **mvn package** находясь в папке **gswm-project**, как отображено в **Листинге 6-7**.

Listing 6-7. Команда **Maven package**, запущенная в директории родительского проекта

```
C:\apress\gswm-book\chapter6\gswm-parent>mvn package
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] gswm-parent
[INFO] gswm-web Maven Webapp
[INFO] gswm-service
[INFO] gswm-repository
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] gswm-parent ..... SUCCESS [0.001s]
[INFO] gswm-web Maven Webapp ..... SUCCESS [1.033s]
[INFO] gswm-service ..... SUCCESS [0.552s]
[INFO] gswm-repository ..... SUCCESS [0.261s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.949s
[INFO] Finished at: Mon Oct 13 23:09:21 MDT 2014
[INFO] Final Memory: 6M/15M
[INFO] -----
```

Создание архетипов

Maven предоставляет несколько способов создать новый архетип. Здесь для этого мы будем использовать уже существующий проект.

Начнем с создания прототипного проекта, который будет использоваться в качестве основы для генерации архетипа. Этот проект будет **Servlet 3.0**-совместимым и содержит **Status Servlet**, возвращающий код **200 HTTP-состояния** («ОК» – успешный запрос). Вместо создания веб-проекта «с нуля» скопируем ранее сгенерированный проект **gswm-web** и создадим **gswm-web-prototype** в папке **C:\apress\gswm-book\chapter6**. Внесите следующие изменения в только что скопированный проект:

1. Удалите все прочие ресурсы, такие как файлы, специфичные для **Integrated Development Environment (IDE)** (**.project**, **.classpath** и т.д.), которые вы не хотите включать в архетип.

2. Замените содержимое файла `web.xml` из папки `webapp/WEB-INF`. Это настроит веб-приложение на использование **Servlet 3.0**:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
<display-name>Archetype Created Web Application</display-name>
</web-app>
```

3. Добавьте зависимость **Servlet 3.0** в файл `pom.xml`. Обновленное содержимое `pom.xml` отображено в **Листинге 6-8**.

Листинг 6-8. Файл `pom.xml` с **Servlet Dependency**

```
<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.apress.gswmbook</groupId>
  <artifactId>gswm-web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>gswm-web Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>gswm-web</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.tomcat.maven</groupId>
```

```

        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
    </plugin>
</plugins>
</build>
</project>

```

4. Т.к. мы будем вести разработку веб-проекта на **Java**, то создайте папку с именем **java** в директории **src/main**. Аналогично создайте папки **test/java** и **test/resources** в директории **src**.
5. Создайте файл **AppStatusServlet.java**, принадлежащий пакету **com.apress.gswmbook.web.servlet** в директории **src/main/java**. Пакет **com.apress.gswmbook.web.servlet** преобразуется в структуру папок **com\apress\gswmbook\web\servlet**. Исходный код файла **AppStatusServlet.java** отображен в **Листинге 6-9**.

Listing 6-9. Исходный код Java-класса *AppStatusServlet*

```

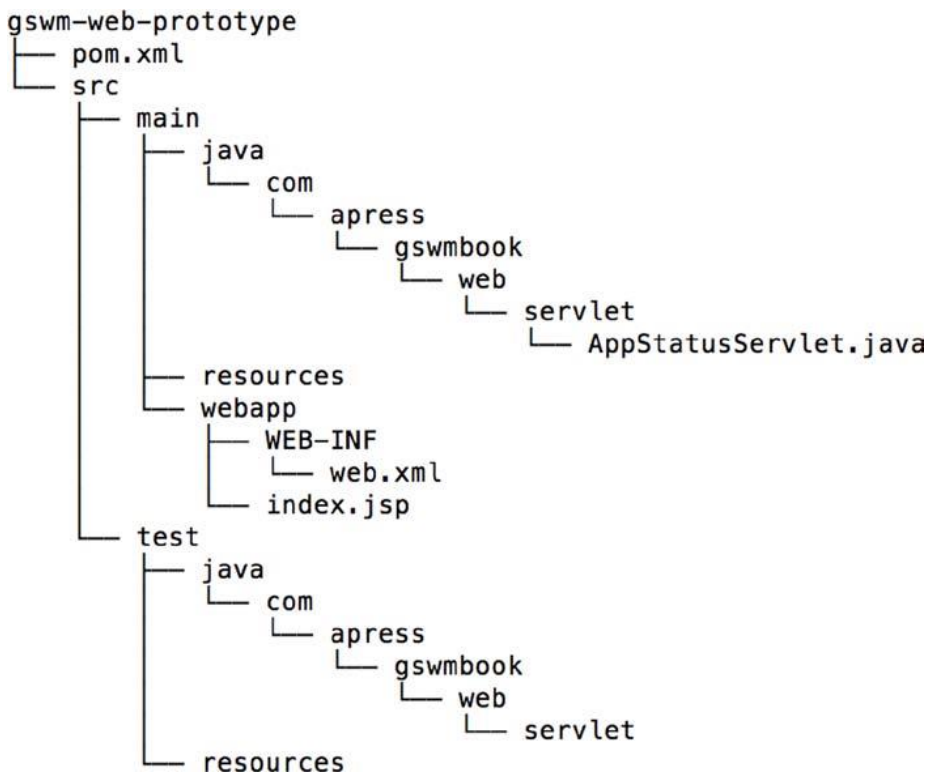
package com.apress.gswmbook.web.servlet;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

@WebServlet("/status")
public class AppStatusServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException {
    PrintWriter writer = response.getWriter();
    writer.println("OK");
    response.setStatus(response.SC_OK);
    }
}

```

Прототипный проект по структуре будет поход на изображенный на **Рис.6-5**.

Рис. 6-5. Сгенерированный прототипный проект



Используя командную строку, перейдите в папку `gswm-web-prototype` проекта и выполните следующую команду:

```
mvn archetype:create-from-project
```

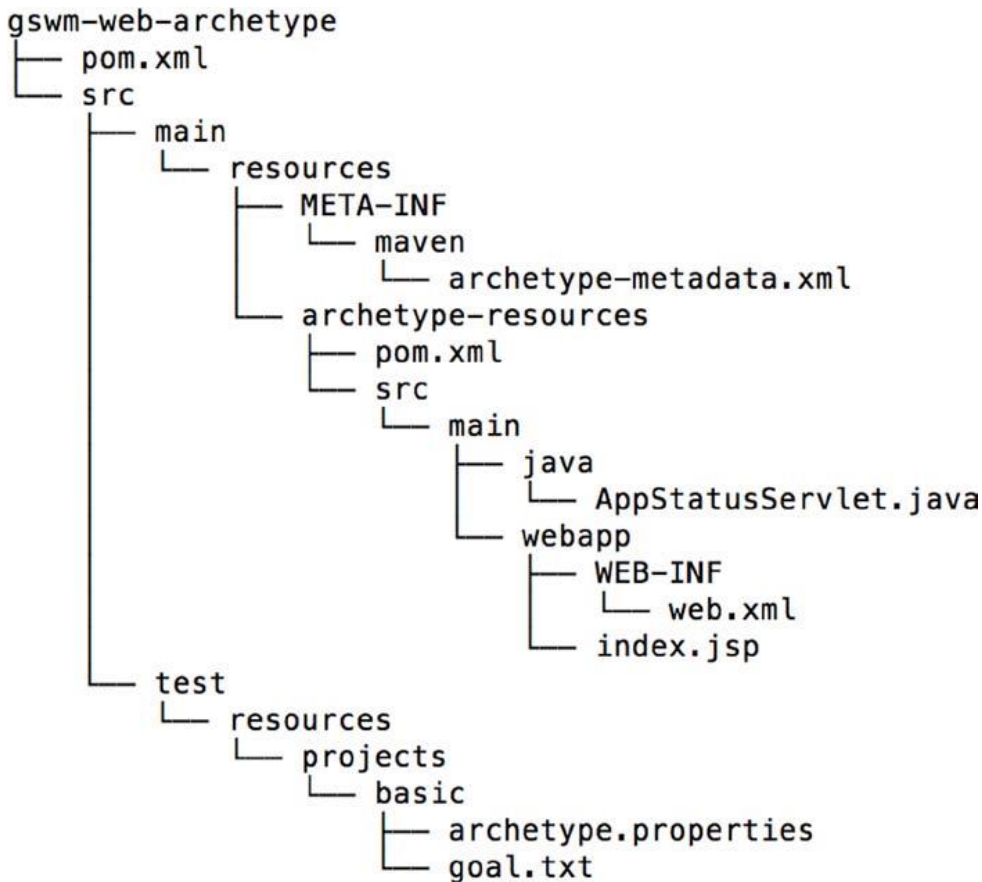
По завершению команды вы увидите сообщение **Archetype created in target/generated-sources/archetype**. Сгенерированный архетип находится в папке `gswm-web-prototype/target/generated-sources/archetype`.

Следующим шагом является перенос только что сгенерированного артефакта в отдельную папку `gswm-web-archetype` для его настройки перед публикацией. Для этого выполните следующие шаги:

1. Создайте папку `gswm-web-archetype` в директории `C:\apress\gswm-book\chapter6`.
2. Скопируйте поддиректории и файлы из папки `C:\apress\gswm-book\chapter6\gswm-web-prototype\target\generated-sources\archetype` в папку `gswm-web-archetype`.
3. Удалите поддиректорию `target` из папки `C:\apress\gswm-book\chapter6\gswm-web-archetype`.

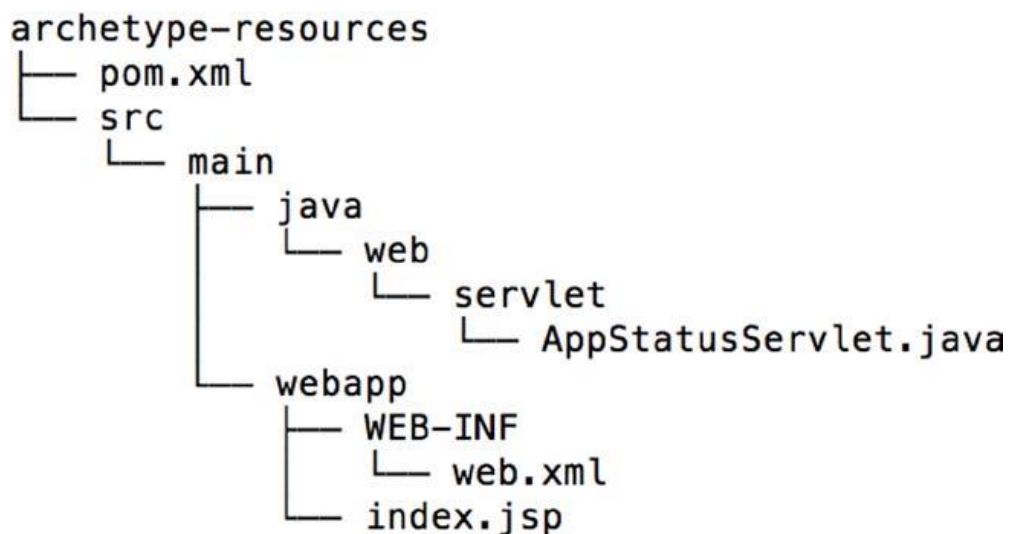
Структура папок для `gswm-web-archetype` должна быть похожа на отображенную на **Рис.6-6**.

Рис. 6-6. Структура проекта веб-архетипа



Давайте начнем процесс модификации с файла `pom.xml`, расположенного в папке `gswm-web-archetype\src\main\resources\archetype-resources`. Измените `<finalName>` в файле `pom.xml` с `gswm-web` на `${artifactId}`. В процессе создания проекта **Maven** заменит выражение `${artifactId}` значением, которое было предоставлено пользователем.

Когда проект создается из архетипа, **Maven** запрашивает у пользователя имя пакета. **Maven** создает структуру папок, соответствующую пакету, находящемуся в директории `src/main/java` созданного проекта. Затем **Maven** перемещает в этот пакет все содержимое из папки `archetype-resources/src/main/java` архетипа. Т.к. вы хотите, чтобы `AppStatusServlet` находился во вложенном пакете `web.servlet`, то создайте папку `web/servlet` и переместите `AppStatusServlet` туда. Новое расположение `AppStatusServlet.java` отображено на Рис. 6-7.

Рис.6-7. AppStatusServlet в пакете web.servlet

Откройте [AppStatusServlet.java](#) и измените имя пакета с `package ${package};` на `package ${package}.web.servlet;`

Завершающим шагом в создании архетипа является выполнение следующей команды из командной строки, находясь в папке [gswm-web-archetype](#):

```
mvn clean install
```

Использование архетипов

Как только архетип был инсталлирован, то простейшим путем создать из него проект – это, находясь в папке [C:\apress\gswm-book\chapter6](#), выполнить следующую команду:

```
mvn archetype:generate -DarchetypeCatalog=local
```

В ответ на запросы **Maven** введите значения, указанные в **Листинге 6-10**, и вы увидите созданный проект.

Listing 6-10. Создание нового проекта с использованием архетипа

```

C:\apress\gswm-book\chapter6>mvn archetype:generate -DarchetypeCatalog=local
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.
maven.archetypes:maven-archetype-quickstart:1.0)

```

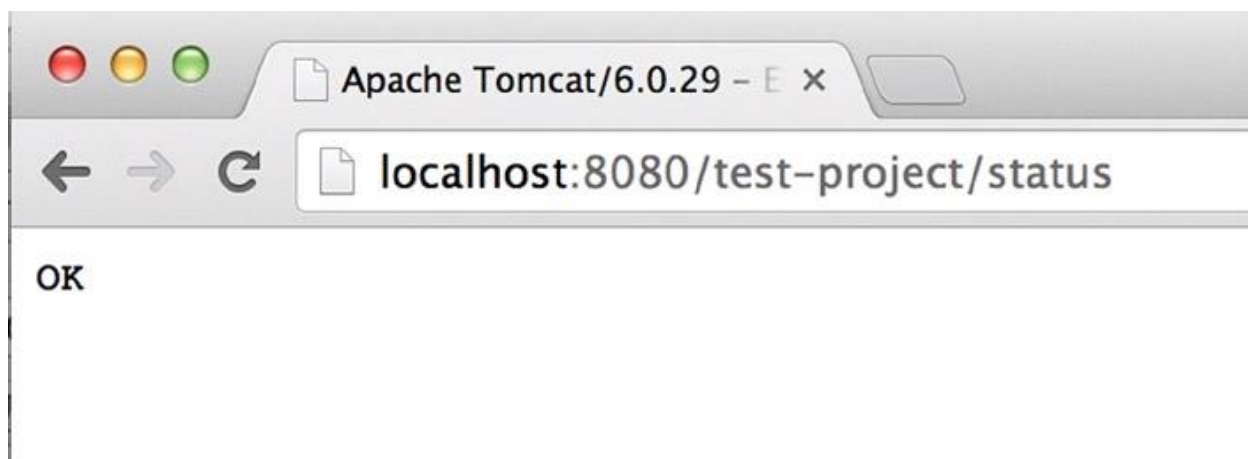
```

Choose archetype:1: local -> com.apress.gswmbook:gswm-web-archetype
(gswm-web-archetype)
Choose a number or apply filter (format: [groupId:]artifactId, case
sensitive contains): : 1
Define value for property 'groupId': : com.apress.gswmbook
Define value for property 'artifactId': : test-project
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': com.apress.gswmbook: :
Confirm properties configuration:
groupId: com.apress.gswmbook
artifactId: test-project
version: 1.0-SNAPSHOT
package: com.apress.gswmbook
Y: :
-----
project
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1:27.635s
[INFO] Finished at: Mon Oct 13 23:36:01 MDT 2014
[INFO] Final Memory: 9M/22M
[INFO] -----

```

Т.к. файл `pom.xml` для **test-project** уже содержит плагин **Tomcat**, то для запуска проекта вызовите команду `mvn tomcat7:run`, находясь в папке `C:\apress\gswmbook\chapter6\test-project`. Откройте браузер и перейдите по адресу <http://localhost:8080/test-project/status>. Вы увидите надпись ОК, как отображено на **Рис.6-8**.

Рис.6-8. Страница, выводимая сгенерированным проектом



Итоги

Архетипы **Maven** являются заготовками проектов, позволяющими быстро запускать новые проекты. В данной главе для генерации сложных **Maven**-проектов, таких как веб-проекты или мультимодульные проекты, использовались встроенные архетипы. Также вы познакомились с созданием и использованием пользовательских архетипов.

В следующей главе вы познакомитесь с основами генерации сайта, создания документации и отчетов при помощи **Maven**.

Глава 7: Документация и отчетность

Документация и отчетность являются ключевыми аспектами любого проекта. Это особенно верно для проектов корпоративного уровня и проектов с открытым исходным кодом, когда над созданием проекта работает множество людей. В этой главе будут рассмотрены некоторые из инструментов и плагинов **Maven**, которые значительно упрощают публикацию и поддержку онлайн-документации.

В этой главе снова будет использоваться Java-проект **gswm**, созданный в предыдущих главах. Проект **gswm** также можно взять из папки <C:\apress\gswm-book\chapter7>.

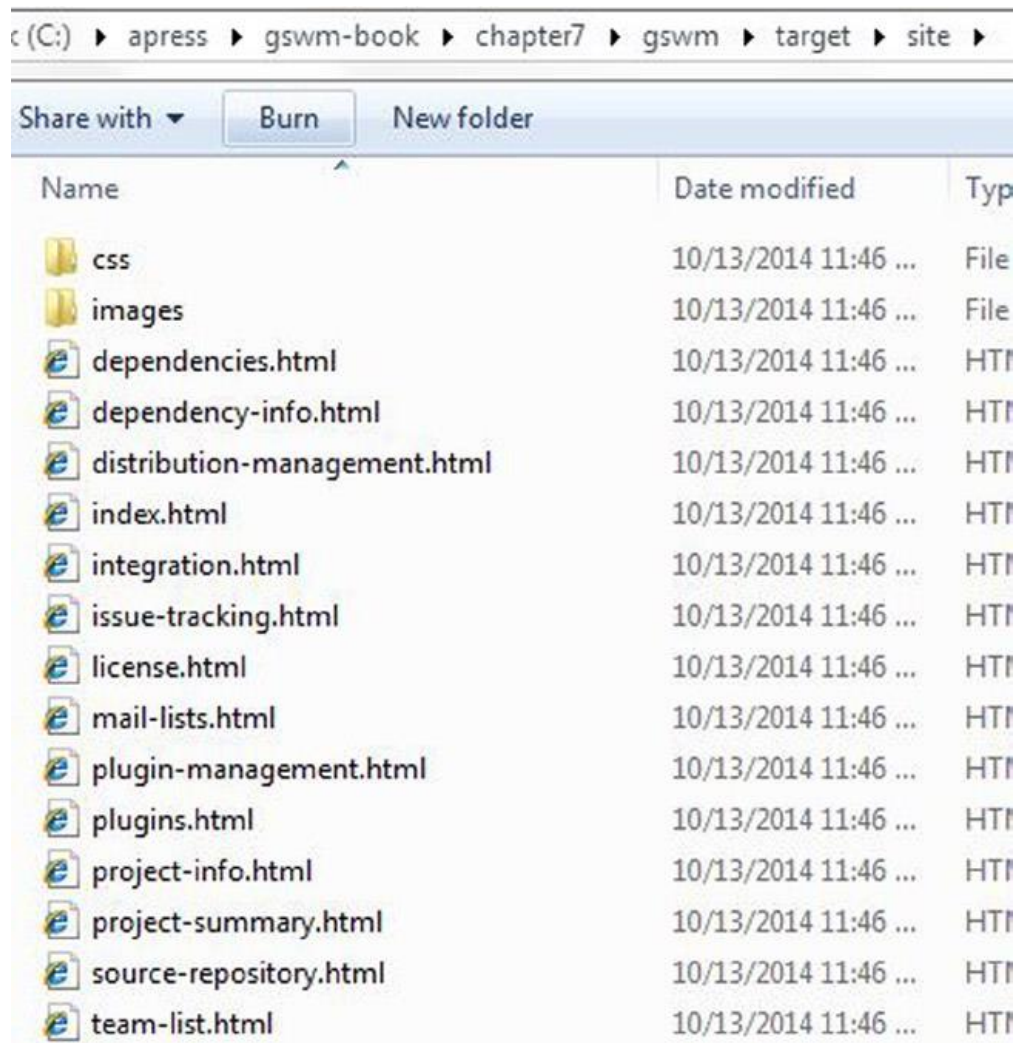
Использование жизненного цикла сайта

Как уже обсуждалось в **Главе 5**, **Maven** обеспечивает *сайту* жизненный цикл, который может быть использован для генерации документации проекта. Давайте выполним следующую команду, находясь в директории **gswm**:

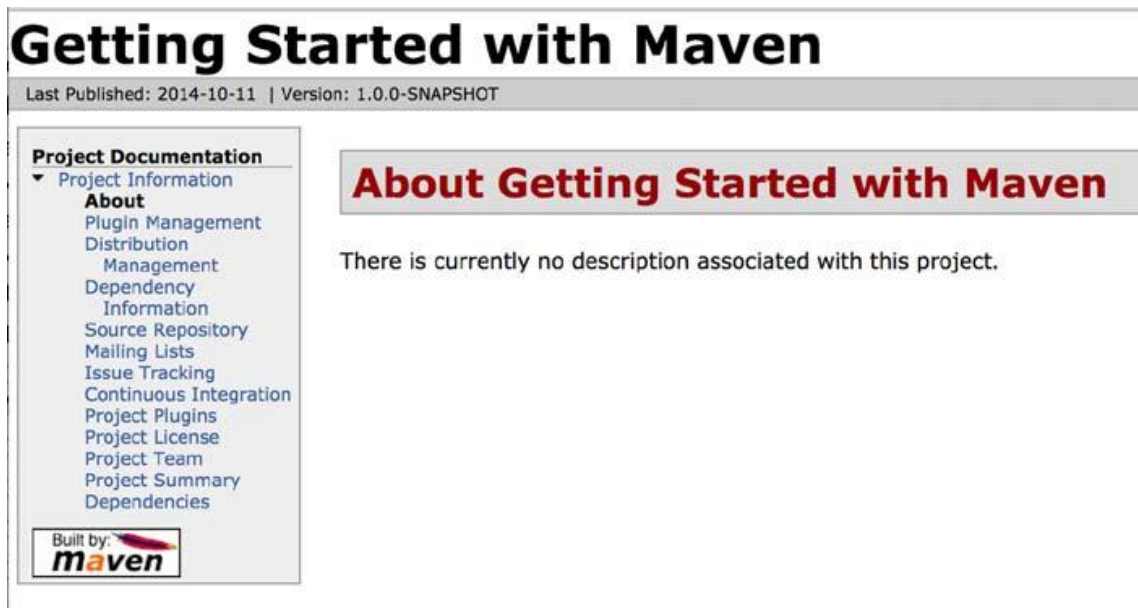
```
mvn site
```

Жизненный цикл сайта использует **Maven**-плагин для генерации сайта проекта. Как только эта команда выполнится, в папке **target** проекта будет создана папка сайта **site**. **Рисунок 7-1** отображает содержимое этой папки.

Рисунок 7-1. Сгенерированная папка [site](#)

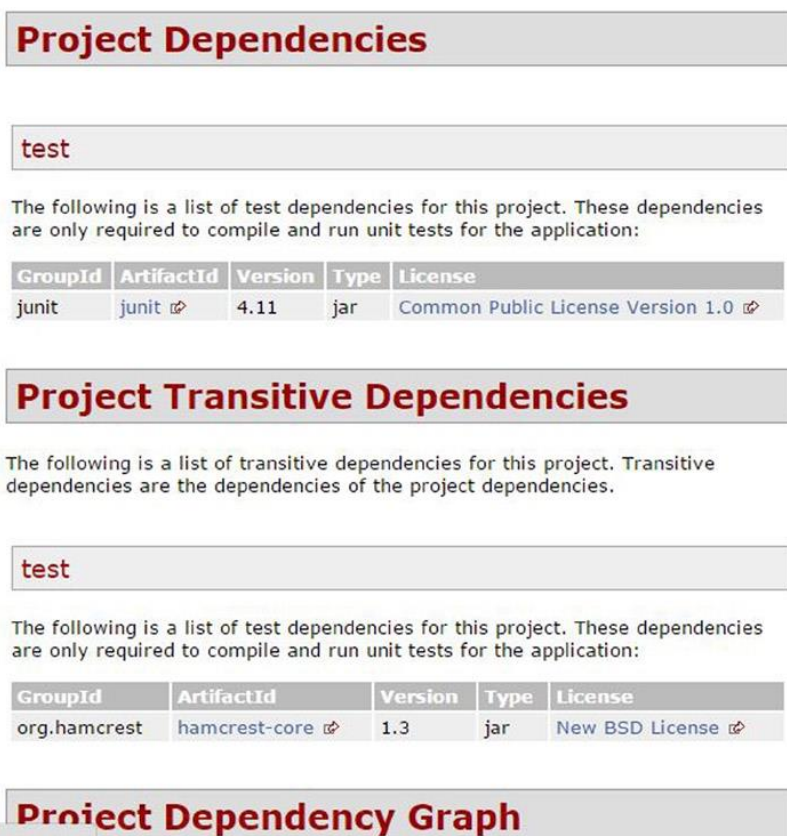


Откройте файл [index.html](#) чтобы увидеть сгенерированный сайт. Можно заметить, что для генерации большинства документации **Maven** использовал информацию из файла [pom.xml](#). **Maven** также автоматически использовал оформление по умолчанию с соответствующими изображениями и **CSS**-файлами. **Рисунок 7-2** отображает сгенерированный файл [index.html](#).

Рисунок 7-2. Сгенерированная начальная страница [index.html](#)


Страница **Project Dependencies** предоставляет важную информацию относительно прямых и транзитивных зависимостей проекта. Также отображена информация о лицензиях этих зависимостей, как отображено на **Рисунке 7-3**.

Рисунок 7-3. Страница зависимостей проекта



Project Dependencies

test

The following is a list of test dependencies for this project. These dependencies are only required to compile and run unit tests for the application:

GroupId	ArtifactId	Version	Type	License
junit	junit	4.11	jar	Common Public License Version 1.0

Project Transitive Dependencies

The following is a list of transitive dependencies for this project. Transitive dependencies are the dependencies of the project dependencies.

test

The following is a list of test dependencies for this project. These dependencies are only required to compile and run unit tests for the application:

GroupId	ArtifactId	Version	Type	License
org.hamcrest	hamcrest-core	1.3	jar	New BSD License

Project Dependency Graph

Просматривая сгенерированный сайт, вы заметите, что на таких страницах, как **About, Mailing Lists** и **Project License** информация отсутствует. Давайте модифицируем файл `pom.xml` и добавим отсутствующую информацию, как отображено на **Листинге 7-1**.

Listing 7-1. Файл `pom.xml`, содержащий информацию о проекте

```
<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.apress.gswmbook</groupId>
  <artifactId>gswm</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Getting Started with Maven</name>
  <url>http://apress.com</url>
  <description>
    This project acts as a starter project for the Introducing Maven book
    (http://www.apress.com/9781484208427) published by Apress.
  </description>
  <mailingLists>
    <mailingList>
      <name>GSWM Developer List</name>
      <subscribe>gswm-dev-subscribe@apress.com</subscribe>
      <unsubscribe>gswm-dev-unsubscribe@apress.com</unsubscribe>
      <post>developer@apress.com</post>
    </mailingList>
  </mailingLists>
  <licenses>
    <license>
      <name>Apache License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
    </license>
  </licenses>
  <!-- Developer, Dependency and Build information removed for brevity -->
</project>
```

Глядя на содержимое файла `pom.xml` в **Листинге 7-1** становится очевидным, что элемент **<description>** используется для указания описания проекта. Элемент **<mailingList>** содержит список адресов электронной почты, а элемент **<license>** - информацию о лицензии проекта. Давайте сгенерируем сайт, выполнив следующую команду:

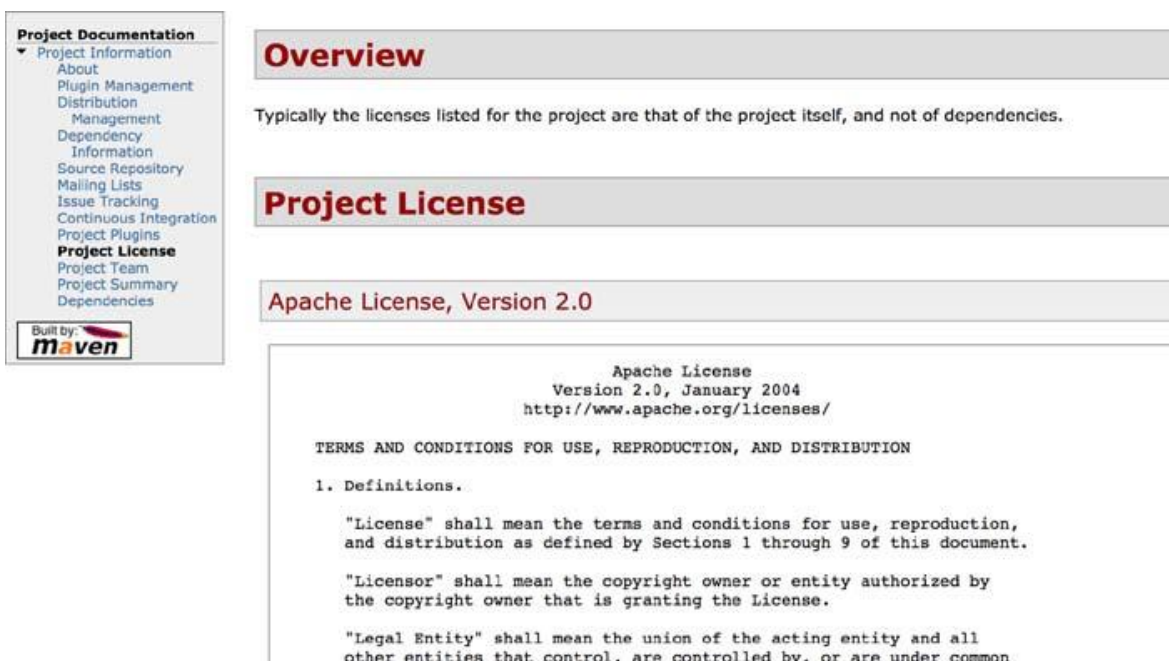
```
mvn clean site
```

Откройте файл [index.html](#) в заново сгенерированной папке [target/site](#). **Рисунки 7-4А и 7-4Б** отображают новые страницы **About** и **Project License** соответственно. Обратите внимание, что **Maven** использует **URL**, объявленный в элементе `<license>`, для скачивания текста лицензии и включает его в сгенерированный веб-сайт.

Рисунок 7-4А. Сгенерированная страница **About**



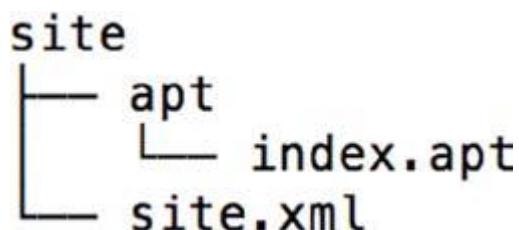
Рисунок 7-4Б. Сгенерированная страница **Project License**



Дополнительное конфигурирование сайта

В предыдущем разделе информация о проекте была указана в файле `pom.xml`, чтобы **Maven** мог ей воспользоваться при генерации сайта. В средних и больших проектах такой подход приводит к возникновению разросшихся и трудных в поддержке файлов `pom.xml`. Кроме того, предприятия обычно предпочитают использовать собственные бренды и логотипы на сгенерированном сайте вместо оформления, предоставляемого **Maven** по умолчанию. Для реализации этого **Maven** позволяет указывать содержимое и конфигурацию сайта в папке `src/site`. **Рисунок 7-5** отображает структуру директорий простой папки `site`.

Рисунок 7-5. Структура папки `site`



Файл `site.xml`, также известный, как *дескриптор сайта*, используется для настройки генерируемого сайта. Этот элемент мы рассмотрим немного позже.

Папка `apt` содержит наполнение сайта, написанное в формате *Almost Plain Text (APT)*. Формат **APT** позволяет создавать документацию, используя синтаксис, подобный простому тексту. Дополнительную информацию о формате **APT** можно получить на веб-сайте **Maven** (<http://maven.apache.org/doxia/references/apt-format.html>). В дополнение к **APT** **Maven** поддерживает другие форматы, такие, как **FML**, **Xdoc** и **Markdown**.

Maven предоставляет несколько архетипов, предоставляющих автоматически сгенерировать структуру сайта. Поскольку мы будем обновлять существующий проект `gswm`, то вместо цели `generate` мы будем использовать цель `create`, как отображено в следующей команде¹. Запустите эту команду, находясь в папке `C:\apress\gswm-book\chapter7\gswm folder`:

```
mvn archetype:create -DarchetypeArtifactId=maven-archetype-site-simple
```

После успешного завершения команды вы увидите папку `site`, созданную внутри папки `gswm/src`, содержащую файл `site.xml` и папку `apt`. Начнем с того, что добавим описание проекта в файл `index.apt`. Замените содержимое файла `index.apt` кодом из **Листинга 7-2**.

¹ В более поздних версиях архетипа цель `create` была удалена и придется всё же использовать цель `generate` для генерации `site.xml` и `index.apt`, а потом вызывать `mvn site` для генерации сайта, использующего настройки, указанные в этих файлах (прим.перев.).

Listing 7-2. Содержимое файла `index apt`

```
-----
Getting Started with Maven Starter
-----
Apress
-----
10-10-2014
-----
```

Этот проект выступает стартовым проектом для книги *Introducing Maven*, выпущенной Apress. Для получения дополнительной информации посетите сайте Apress <https://www.apress.com>.

Рисунок 7-6. Страница **About** с обновлённым содержимым



Обратите внимание, что левая навигационная панель полностью исчезла. Причина в том, что **Maven** создает эту панель, используя файл `site.xml`, а в данный момент в файле `site.xml` навигационная информация отсутствует.

Перед тем, как рассмотреть содержимое файла `site.xml`, давайте добавим изображение, служащее логотипом сайта. Статические активы, такие как изображения и HTML-файлы, размещаются в папке `site/resources`. Когда **Maven** генерирует сайт, он копирует эти активы в папку `resources`, находящуюся в корне сгенерированного сайта. Скопируйте логотип компании `company.png` из папки `C:\apress\gswm-book\chapter7` в папку `gswm/src/site/resources/images`.

Теперь можно модифицировать файл `site.xml` для того, чтобы появились логотип и панель навигации. Заполните файл `site.xml` содержимым **Листинга 7-3**. Обратите внимание, что элемент `<src>` для логотипа использует относительный путь `images/company.png`. Элемент `<menu>` используется для создания различных навигационных ссылок, отображаемых на сайте.

Listing 7-3. The `site.xml` File Contents

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project xmlns="http://maven.apache.org/DECORATION/1.6.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/DECORATION/1.6.0
http://maven.apache.org/xsd/decoration-1.6.0.xsd"
name="Getting Started With Maven">
```

```

<bannerLeft>
  <name>Apress</name>
  <src>images/company.png</src>
  <href>http://apress.com</href>
</bannerLeft>
<body>
  <links>
    <item name="Maven" href="http://maven.apache.org/" />
  </links>
  <menu name="Project Information">
    <item name="Introduction" href="index.html" />
    <item name="Plugin Management" href="plugin-management.html" />
    <item name="Dependency Information" href="dependency-info.html" />
    <item name="Source Repository" href="source-repository.html" />
    <item name="Mailing Lists" href="mail-lists.html" />
    <item name="Issue Tracking" href="issue-tracking.html" />
    <item name="Continuous Integration" href="integration.html" />
    <item name="Project Plugins" href="plugins.html" />
    <item name="Project License" href="license.html" />
    <item name="Project Team" href="team-list.html" />
    <item name="Project Summary" href="project-summary.html" />
    <item name="Dependencies" href="dependencies.html" />
  </menu>
  <menu name="Reports">
  </menu>
</body>
</project>

```

Выполнение команды `mvn clean site` сгенерирует сайт с новым логотипом и навигационной панелью, как отображено на **Рисунке 7-7**.

Рисунок 7-7. Страница **About** с новым логотипом



Генерация отчетов Javadoc

Javadoc де-факто является стандартом для документирования **Java**-кода. Он помогает разработчикам понимать, что делает метод или класс. **Javadoc** также выделяет устаревшие поля, методы или классы.

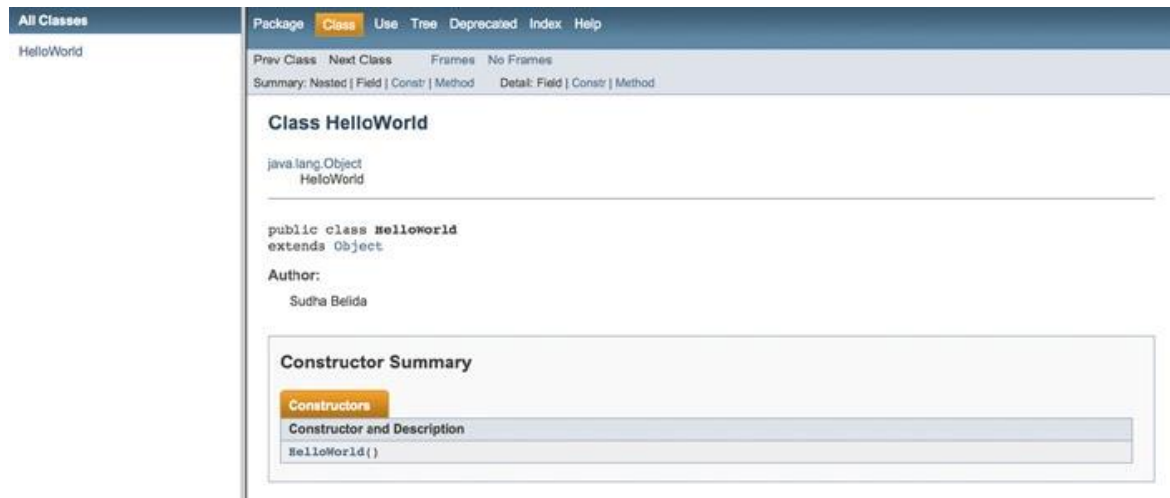
Maven предоставляет плагин для **Javadoc**, использующий инструмент **Javadoc** для генерации описаний. Интеграция плагина **Javadoc** состоит просто в его объявлении в элементе `<reporting>` в файле `pom.xml`, как это отображено в **Листинге 7-4**. Плагины, объявленные в элементе `<reporting>` файла `pom.xml`, исполняются в течении генерации сайта.

Listing 7-4. Объявление плагина Javadoc в файле pom.xml

```
<project>
  <!--Content removed for brevity-->
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.10.1</version>
      </plugin>
    </plugins>
  </reporting>
</project>
```

Теперь, когда у нас есть сконфигурированный плагин **Javadoc**, давайте выполним команду `mvn clean site` для генерации **Javadoc**. После успешного выполнения команды вы заметите, что в директории `gswm/target/site` появилась папка `apidocs`. Двойным щелчком по файлу `index.html`, находящемуся в папке `apidocs`, вы откроете **Javadoc**. **Рисунок 7-8** отображает **Javadoc**, сгенерированный для проекта `gswm`.

Рисунок 7-8. Сгенерированная страница Javadoc



Генерация отчетов юнит-тестирования

Разработка через тестирование (**TDD, Test-driven development**) стала нормой в наши дни. **Юнит-тесты** обеспечивают разработчикам немедленный отклик и позволяют разрабатывать качественный код. Принимая во внимание важность тестов, **Maven** выполняет их для каждой сборки. Неудача любого из тестов приводит и к неудаче всей сборки.

Maven предоставляет плагин **Surefire**, обеспечивающий унифицированный интерфейс для запуска тестов, созданный такими фреймворками, как **JUnit** или **TestNG**. Также он генерирует результаты исполнения в различных форматах, таких, как **XML** и **HTML**. Эти публикуемые результаты позволяют разработчикам находить и быстро исправлять неуспешные тесты.

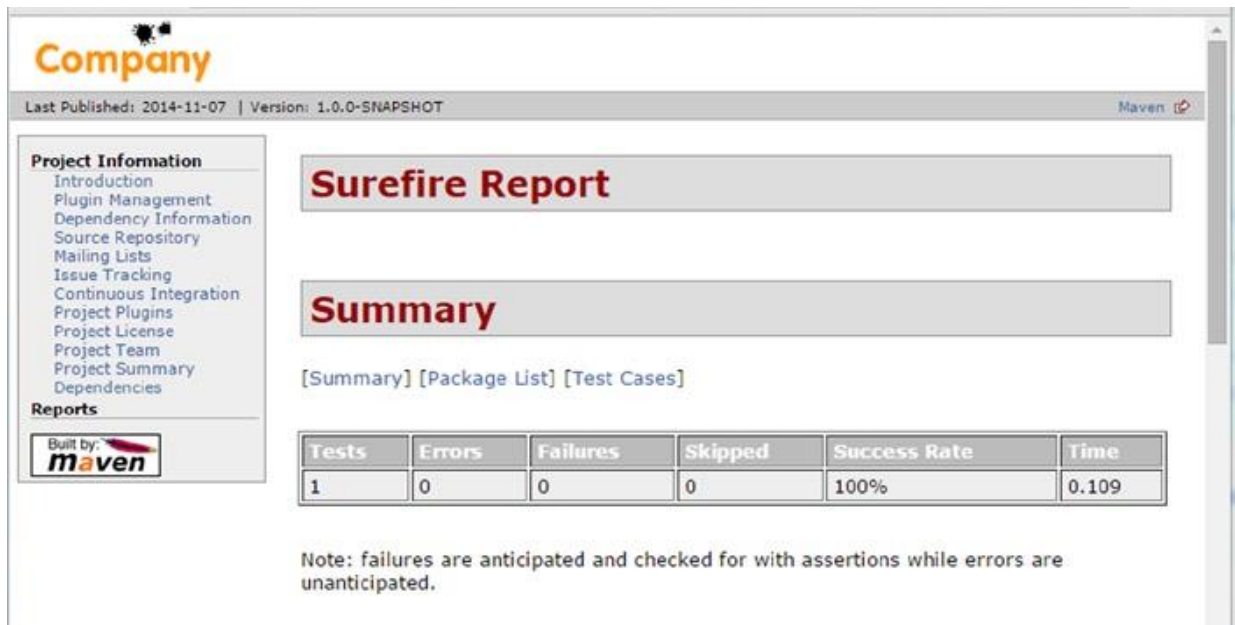
Плагин **Surefire** сконфигурирован таким же образом, как плагин **Javadoc** в секции **<reporting>** файла **pom.xml**. **Листинг 7-5** отображает конфигурацию плагина **Surefire**.

Листинг 7-5. Фрагмент файла **pom.xml** с конфигурацией плагина **Surefire**

```
<project>
  <!--Content removed for brevity-->
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-report-plugin</artifactId>
        <version>2.17</version>
      </plugin>
    </plugins>
  </reporting>
</project>
```

Теперь, когда **Surefire** сконфигурирован, сгенерируем сайт путем запуска команды `mvn clean site`. После успешного выполнения команды вы увидите папку `surefire-reports`, созданную в директории `gswm/target`. Она содержит результаты исполнения теста в форматах **XML** и **TXT**. Та же самая информация будет доступна в **HTML**-формате в файле `surefire-report.html`, находящегося в папке `site`. **Рисунок 7-9** отображает **Surefire Report** для проекта `gswm`.

Рисунок 7-9. Сгенерированный отчет **Surefire**



Генерация отчетов о покрытии кода

Покрытие кода является мерой того, какая доля исходного кода проверяется автоматическими тестами. По существу, он является показателем качества ваших тестов. **Emma** и **Cobertura** – это два популярных инструмента с открытым исходным кодом для **Java**, позволяющие рассчитать процент покрытия кода тестами.

В этом разделе вы будете использовать **Cobertura** для измерения покрытия кода этого проекта. Конфигурирование **Cobertura** подобно конфигурированию прочих плагинов, как демонстрируется в **Листинге 7-6**.

Listing 7-6. Отрывок файла `pom.xml` с плагином **Cobertura**

```
<project>
  <!--Content removed for brevity-->
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
```

```

        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.6</version>
    </plugin>
</plugins>
</reporting>
</project>

```

Теперь, когда плагин сконфигурирован, сгенерируем сайт с помощью команды `mvn clean site`. После успешного завершения этой команды **Cobertura** создаст папку `cobertura` в директории `gswm/target/site`. Откройте отчет двойным щелчком по файлу `index.html`. Отчет должен быть похож на изображенный на **Рисунке 7-10**.

Рисунок 7-10. Сгенерированный отчет Cobertura

Package	# Classes	Line Coverage	Branch Coverage	Complexity
(default)	1	100% 3/3	N/A	1
All Packages	1	100% 3/3	N/A	1
Classes in this Package		Line Coverage	Branch Coverage	Complexity
HelloWorld		100% 3/3	N/A	1

Report generated by Cobertura 2.0.3 on 10/11/14 8:35 PM.

Генерация отчета FindBugs

FindBugs является инструментом для поиска дефектов в **Java**-коде. Он использует статистический анализ для обнаружения ошибочных шаблонов, таких, как бесконечный рекурсивный цикл и **null**-разыменования. **Листинг 7-7** отображает конфигурацию **FindBugs**.

Listing 7-7. Отрывок файла `pom.xml` с плагином **FindBugs**

```

<project>
<!--Content removed for brevity-->
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>findbugs-maven-plugin</artifactId>
        <version>3.0.0</version>
      </plugin>
    </plugins>

```

```
</reporting>
</project>
```

Как только сайт **Maven** будет сгенерирован, запустите отчет **FindBugs**, открыв файл [findbugs.html](#), располагающийся в директории `C:\apress\gswm-book\chapter7\gswm\target\site`. Он должен быть похож на изображенный на **Рисунке 7-11**.

Figure 7-11. Сгенерированный отчет обнаружения ошибок **FindBugs**

The screenshot displays a web browser window showing a Maven-generated report. The page title is "Company" and it includes a navigation menu on the left with options like "Project Information", "Reports", and "Built by: maven". The main content area is titled "FindBugs Bug Detector Report" and contains the following text:

The following document contains the results of FindBugs

FindBugs Version is 3.0.0

Threshold is *medium*

Effort is *min*

Below this text is a "Summary" section with a table:

Classes	Bugs	Errors	Missing Classes
1	0	0	0

At the bottom of the report, there is a section titled "Files".

Итоги

Возможности документирования и отчетности, предоставляемые **Maven**, играют важную роль при создании качественных и удобных в поддержке программ. В этой главе объяснялись основы использования жизненного цикла сайта и конфигурирования, необходимого для создания документации. Также вы рассмотрели генерацию **Javadoc**, покрытие тестами и отчеты **FindBugs**.

В следующей главе мы объясним, как интегрировать **Maven** с **Nexus** и **SVN**. Также вы узнаете о процессе публикации **Maven**.

Глава 8: Публикация с помощью Maven

Интеграция с Nexus

Менеджеры хранилищ (репозиториев) являются ключевым элементом развертывания **Maven** на предприятиях. *Менеджеры репозиториев* выступают в качестве представителей публичных репозиториев, облегчая обмен артефактами и командное взаимодействие, гарантируют стабильность сборки и реализуют управление артефактами, используемое на предприятии.

Nexus является популярным менеджером с открытым исходным кодом от **Sonatype**. Это веб-приложение, позволяющее поддерживать внутренние репозитории и доступ к внешним репозиториям. Он позволяет группировать репозитории и получать к ним доступ через общий **URL**. Это позволяет администраторам репозиториев неявно добавлять и удалять новые репозитории, не требуя от разработчиков изменять конфигурации на их компьютерах. Дополнительно, они обеспечивают возможность хостинга для сайтов, сгенерированных **Maven**, и функции поиска артефактов.

Перед рассмотрением интеграции **Maven** и **Nexus**, необходимо установить **Nexus** на ваш локальный компьютер. **Nexus** распространяется в виде архива и поставляется в связке с экземпляром **Jetty**. Скачайте дистрибутив **Nexus** (.zip-версия для Windows) с веб-сайта **Sonatype** www.sonatype.com/download-oss-sonatype. На момент написания данной книги для **Nexus** была доступна версия **2.10.0-02**². Распакуйте содержимое архива на ваш компьютер. В данной книге мы подразумеваем, что содержимое окажется в папке [C:\tools\nexus folder](#).

Замечание: Большинство предприятий обычно устанавливают менеджеры репозиториев на центральный сервер. Если у вас уже есть доступ к менеджеру репозиториев, то пропустите эту часть установки.

Откройте командную строку в режиме администратора и перейдите в папку **bin**, расположенную в директории [C:\tools\nexus\nexus-2.10.0-02](#). После этого выполните команду `nexus install`. Вы увидите команду успешной установки, как отображено на **Рисунке 8-1**. Эта команда установит нативную оболочку сервиса, позволяющую **Jetty** запуститься.

² На момент перевода – версия 3.13.0-01.

Рисунок 8-1. Сообщение об успешной инсталляции **Nexus**

```
C:\tools\nexus\nexus-2.10.0-02\bin>nexus install
wrapper ! nexus installed.
```

Замечание: Nexus 2.10 для правильной работы требует **JRE 1.7**. Убедитесь, что на вашей локальной машине установлена **JDK/JRE** версии **1.7**. Также проверьте, чтобы переменная окружения **JAVA_HOME** указывала на **JDK** версии **1.7**.

В этой же командной строке выполните команду `nexus start` для запуска **Nexus**. **Рисунок 8-2** отображает результат выполнения этой команды.

Рисунок 8-2. Запуск **Nexus**

```
C:\tools\nexus\nexus-2.10.0-02\bin>nexus start
wrapper ! Starting the nexus service...
wrapper ! Waiting to start...
wrapper ! Waiting to start...
wrapper ! Waiting to start...
wrapper ! nexus started.
```

По умолчанию **Nexus** запускается на 8081 порту. Откройте веб-браузер и перейдите в **Nexus** по адресу <http://localhost:8081/nexus>. **Рисунок 8-3** отображает экран запуска **Nexus**. Залогиньтесь в **Nexus** с логином **admin** и паролем **admin123**.

Рисунок 8-3. Экран запуска **Nexus**

Теперь, когда Nexus установлен, модифицируем проект `gswm`, расположенный в папке `C:\apress\gswm-book\chapter8`. Начнем с добавления в файл `pom.xml` элемента `<distributionManagement>`, как указано в **Листинге 8-1**. Этот элемент используется для указания расположения, где будут находиться развернутые артефакты проекта. Элемент `<repository>` задает расположение, где будут находиться развернутые выпущенные артефакты. Аналогичным образом, элемент `<snapshotRepository>` указывает расположение, в которое будут сохраняться **SNAPSHOT**-версии проекта.

Листинг 8-1. Файл `pom.xml` с элементом `<distributionManagement>`

```
<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>
  <!-- Content removed for brevity -->
  <distributionManagement>
    <repository>
      <id>nexusReleases</id>
      <name>Releases</name>
      <url>http://localhost:8081/nexus/content/repositories/releases</url>
    </repository>
    <snapshotRepository>
      <id>nexusSnapshots</id>
      <name>Snapshots</name>
      <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
  </distributionManagement>
  <!-- Content removed for brevity -->
</project>
```

Замечание: Nexus поставляется с репозиториями **Releases** и **Snapshots**. По умолчанию **SNAPSHOT**-артефакты будут сохраняться в репозитории **Snapshots**, артефакты релизов будут сохраняться в **Releases**.

Подобно большинству менеджеров репозиторияев разворачивание в **Nexus** является защищенной операцией. В файле `settings.xml` вы указываете учетные данные, необходимые для взаимодействия с **Nexus**.

Листинг 8-2 демонстрирует файл `settings.xml` с серверной информацией. По умолчанию в **Nexus** присутствует пользователь **deployment** с паролем **deployment123**. Обратите внимание, что содержимое тега `<id>`, объявленного в теге `<server>` - **nexusReleases** и **nexusSnapshots** должны совпадать с `<id>` внутри тегов `<repository>` и `<snapshotRepository>`, объявленных в файле `pom.xml`. Замените содержимое файла `settings.xml`, находящегося в папке `C:\Users\<<USER_NAME>>\.m2`, кодом из **Листинга 8-2**.

Листинг 8-2. Файл `settings.xml` с настройками сервера

```
<?xml version="1.0" encoding="UTF-8"?>
<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>nexusReleases</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
    <server>
      <id>nexusSnapshots</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
  </servers>
</settings>
```

Конфигурирование взаимодействия с **Nexus** на этом завершается. В командной строке, находясь в директории `C:\apress\gswm-book\chapter8\gswm`, вызовите команду `mvn deploy`. После успешного выполнения этой команды вы увидите **SNAPSHOT**-артефакт в **Nexus** по адресу <http://localhost:8081/nexus/content/repositories/snapshots/com/apress/gswmbook/gswm/1.0.0-SNAPSHOT/>, как отображено на **Рисунке 8-4**.

Рисунок 8-4. **SNAPSHOT**-артефакт в **Nexus**

Index of /repositories/snapshots/com/apress/gswmbook/gswm/1.0.0-SNAPSHOT

Name	Last Modified	Size	Description
Parent Directory			
gswm-1.0.0-20141015.001443-1.jar	Tue Oct 14 18:14:43 MDT 2014	2382	
gswm-1.0.0-20141015.001443-1.jar.md5	Tue Oct 14 18:14:43 MDT 2014	32	
gswm-1.0.0-20141015.001443-1.jar.sha1	Tue Oct 14 18:14:43 MDT 2014	40	
gswm-1.0.0-20141015.001443-1.pom	Tue Oct 14 18:14:43 MDT 2014	2108	
gswm-1.0.0-20141015.001443-1.pom.md5	Tue Oct 14 18:14:43 MDT 2014	32	
gswm-1.0.0-20141015.001443-1.pom.sha1	Tue Oct 14 18:14:43 MDT 2014	40	
maven-metadata.xml	Tue Oct 14 18:14:44 MDT 2014	773	
maven-metadata.xml.md5	Tue Oct 14 18:14:44 MDT 2014	32	
maven-metadata.xml.sha1	Tue Oct 14 18:14:44 MDT 2014	40	

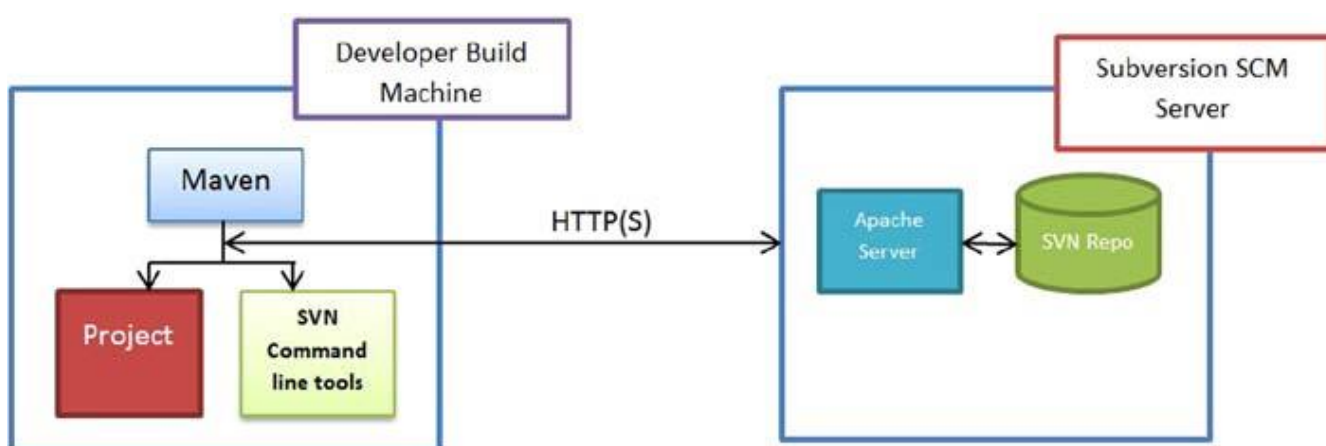
Релиз проекта

Релиз проекта является сложным процессом и обычно состоит из следующих шагов:

- проверка, что все изменения на локальной машине находятся в коммитах;
- удаление **SNAPSHOT** из версии в файле [pom.xml](#);
- проверка того, что проект не использует никаких **SNAPSHOT** зависимостей;
- регистрация модифицированного [pom.xml](#) в системе контроля версий;
- создание тега в системе контроля версий;
- сборка новой версии артефакта и разворачивание его в менеджере репозитория;
- приращение номера версии в [pom.xml](#) и подготовка к новому циклу разработки.

Maven обладает плагином для создания релиза, предоставляющего стандартный механизм для выполнения вышеуказанных шагов и выпуска артефакта проекта. Как можно видеть, в процессе выпуска **Maven** плотно взаимодействует с системой контроля версий. В этом разделе в качестве системы контроля версий мы будем использовать **Subversion (SVN)**. Типичное взаимодействие между **Maven** и **SVN** отображено на **Рисунке 8-5**. **SVN**-сервер управляет репозиториями, содержащими проекты предприятия. Релизы **Maven** обычно выполняются на компьютере разработчика или на сервере сборки. **Maven** требует инструмента командной строки **SVN** для инсталляции на таких машинах. Инструмент командной строки **SVN** позволяет **Maven** взаимодействовать с **SVN** и выполнять такие операции, как проверка кода, создание тегов и т.д.

Рисунок 8-5. Взаимодействие между **Maven** и **Subversion**



Перед тем, как мы глубже погрузимся в процесс релиза с помощью **Maven**, вам нужно подготовить свой локальный компьютер, выполнив следующие шаги:

1. установить на свою машину сервер **Subversion** и инструмент командной строки **SVN**;
2. создать репозиторий **Subversion**;

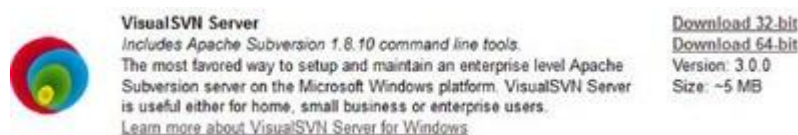
3. проверить проект, который будет использован в репозитории.

Установка инструмента командной строки Subversion

Существует несколько проектов **SVN**-серверов с открытым исходным кодом, предоставляемых коммерческими компаниями. В этом проекте мы будем использовать сервер **Subversion** от **VisualSVN**.

Начните процесс установки со скачивания исполняемого файла 64-битного сервера **VisualSVN** по ссылке www.visualsvn.com/downloads/. Как видно из **Рисунка 8-6**, исполняемые файлы сервера поставляются в комплекте с инструментом командной строки **SVN**.

Рисунок 8-6. Загрузка сервера **VisualSVN**



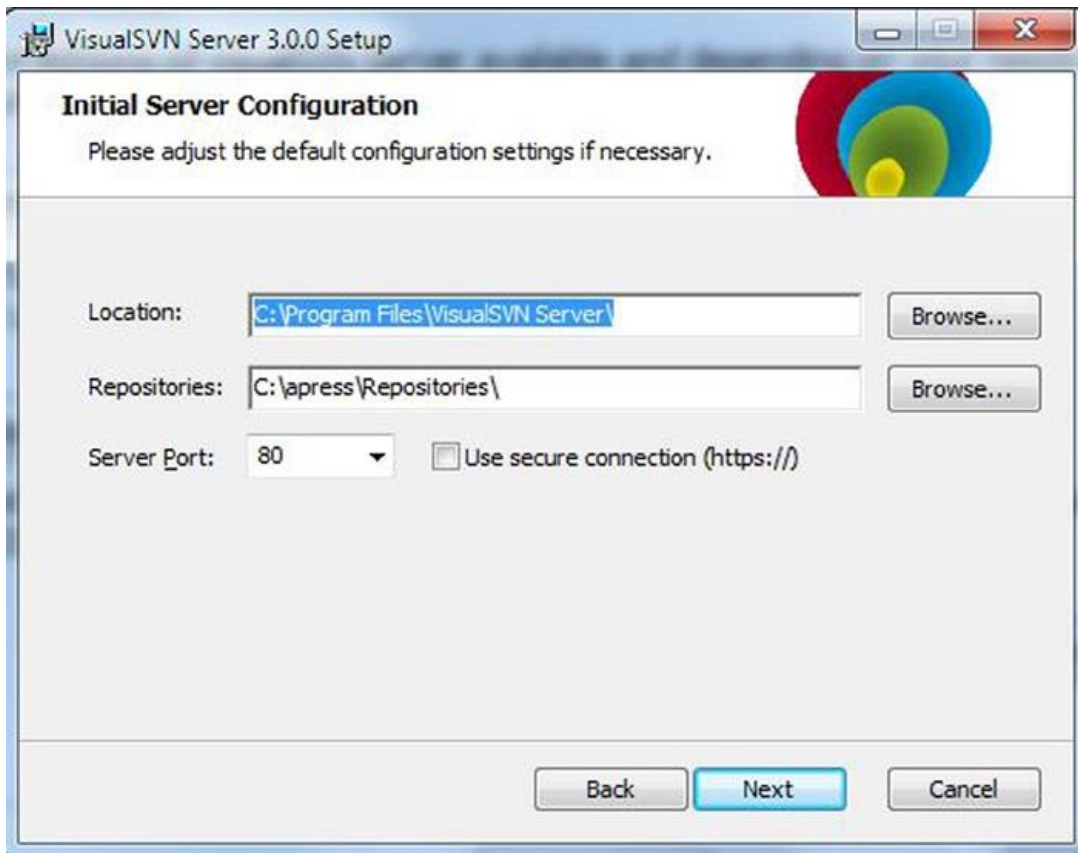
Замечание: Предприятия обычно устанавливают **Subversion** на доступный для использования центральный сервер. Если у вас уже есть доступ к серверу **Subversion**, то можете пропустить шаги по его установке. Однако, вам необходимо иметь инструмент командной строки **SVN** на компьютере, где вы осуществляете релиз с помощью **Maven**. Мы рекомендуем «Инструмент командной строки Apache Subversion» от VisualSVN, который можно скачать и установить по адресу www.visualsvn.com/downloads/.

После скачивания двойным кликом по исполняемому файлу **VisualSVN-Server-3.0.0-x64.exe** откройте экран установки. Примите пользовательское соглашение и на следующем экране установите флажки на опциях «**VisualSVN Server and Management Console**» и «**Add Subversion command-line tools to the PATH environment variable**» как указано на **Рисунке 8-7**.

Рисунок 8-7. Установка VisualSVN

Сервер **VisualSVN** поставляется в двух вариантах: **Standard Edition** и **Enterprise Edition**. Функционала, предоставляемого **Standard Edition** будет вполне достаточно для потребностей данной главы, поэтому кликните по кнопке **Standard Edition**. На следующей строки, снимите флажок «**Use secure connection**», как отображено на **Рисунке 8-8**.

Рисунок 8.8. Пути инсталляции и репозитория



На следующем экране нажмите кнопку **«Install»** для начала инсталляции. После успешной установки **SVN** сервера убедитесь, что инструмент командной строки **SVN** установлен корректно. Для этого откройте новое окно командной строки и выполните команду `svn help`. Вы должны увидеть вывод, похожий на **Рисунок 8-9**.

Рисунок 8-9. Вывод после выполнения команды `svn help`

```

C:\>svn help
usage: svn <subcommand> [options] [args]
Subversion command-line client, version 1.8.10.
Type 'svn help <subcommand>' for help on a specific subcommand.
Type 'svn --version' to see the program version and RA modules
or 'svn --version --quiet' to see just the version number.

Most subcommands take file and/or directory arguments, recursing
on the directories. If no arguments are supplied to such a
command, it recurses on the current directory (inclusive) by default.

Available subcommands:
  add
  blame (praise, annotate, ann)
  cat
  changelist (cl)
  checkout (co)
  cleanup
  commit (ci)
  copy (cp)
  delete (del, remove, rm)

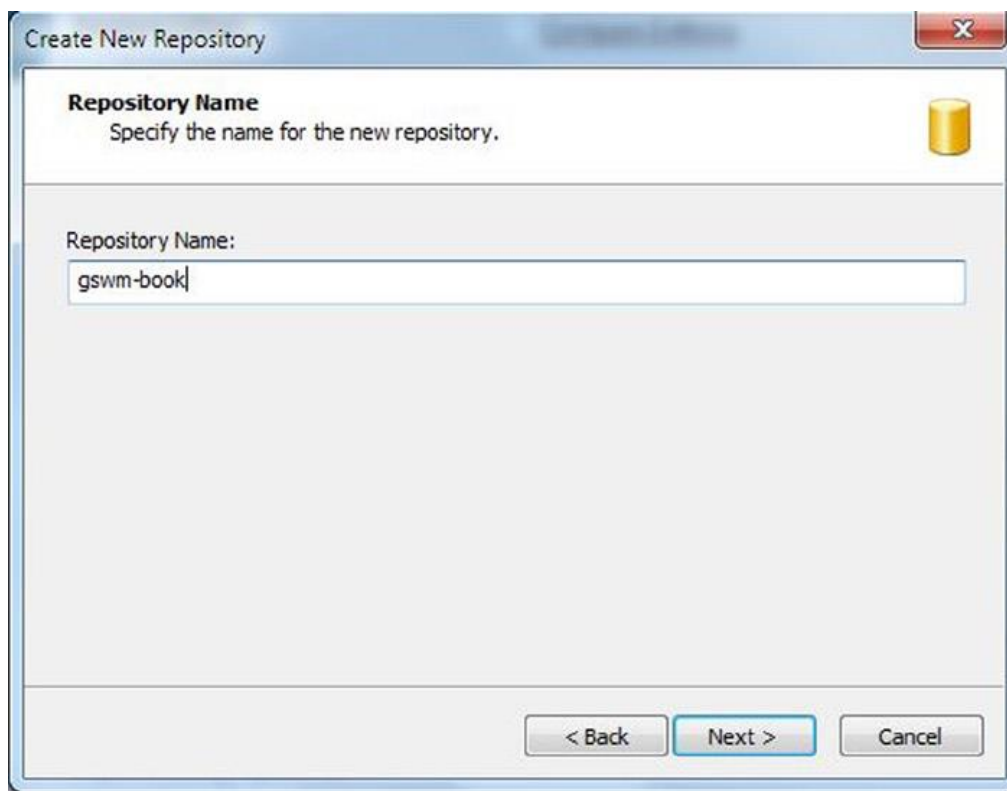
```

Создание репозитория

Репозитории **Subversion** используются для управления файлами и папками и отслеживают любые сделанные в них изменения. **VisualSVN** предоставляет инструмент с замечательным графическим интерфейсом, называемый **VisualSVN Server Manager**, который предельно упрощает создание и управление репозиториями. На **ОС Windows** зайдите в **All Programs ► VisualSVN** и запустите **VisualSVN Server Manager**. Выполните следующие шаги для создания нового репозитория:

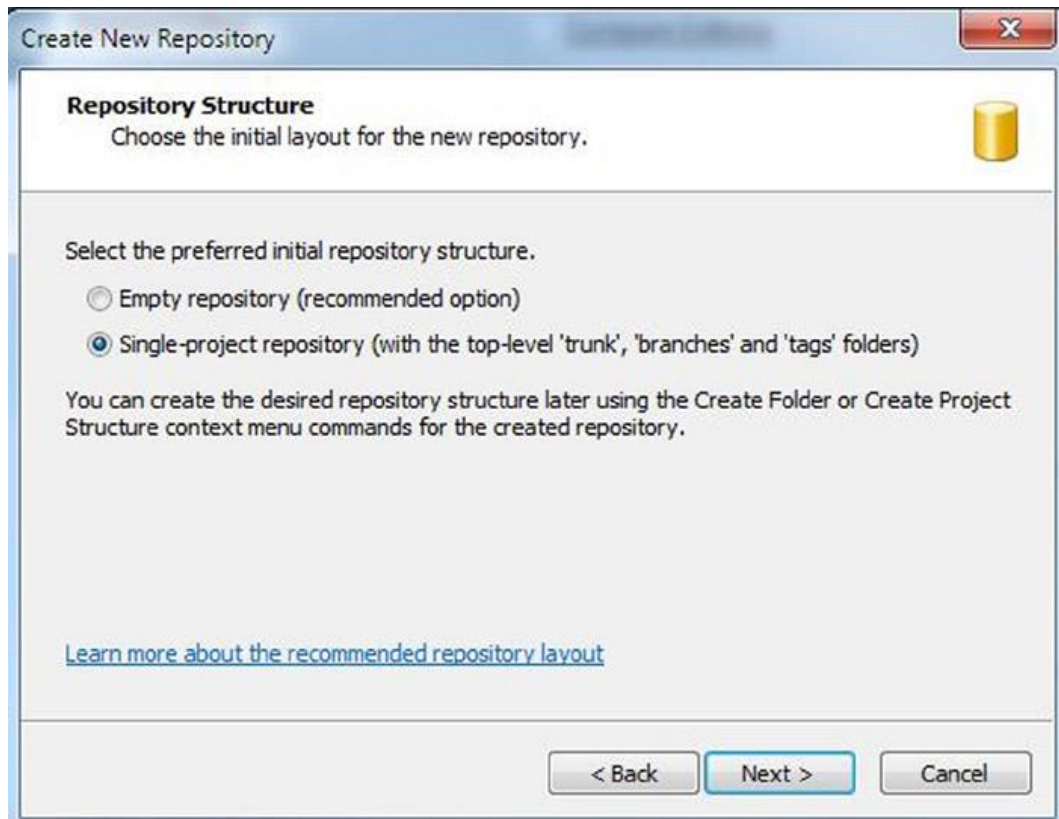
1. В **Server Manager** в секции **Repositories** нажмите **Create a new repository**;
2. На экране **Repository Type** опцию **Regular FSFS** оставьте выбранной. Нажмите **Next**;
3. На **Рисунке 8-10** введите `gswm` в качестве имени репозитория и нажмите **Next**;

Рисунок 8-10. Указание имени репозитория



4. На экране **Repository Structure** выберите опцию **Single project repository**, как показано на **Рисунке 8-11**. Нажмите **Next**;

Рисунок 8-11. Указание структуры репозитория



5. На экране **Repository Access Permissions** оставьте по умолчанию «**All Subversion users have Read/Write access**» и нажмите **Create**. Должно отобразиться сообщение «**A Repository Created Successfully**». Нажмите **Finish**.

Последним шагом в подготовке репозитория является создание нового пользователя, имеющего доступ на чтение и запись к репозиторию **gswm-book**. Для создания такого пользователя выполните следующие шаги:

1. На начальном экране **VisualSVN Service Manager** нажмите **Create a new user** в разделе Subversion Authentication, как указано на Рисунке 8-12.

Рисунок 8-12. Меню создания нового пользователя

Subversion Authentication
 There are 0 users and 0 groups.
[Create new user...](#)
[Create new group...](#)
[Configure authentication options...](#)

2. В окне **Create New User** введите **gswm** в качестве имени пользователя и **gswm** в качестве пароля, как указано на **Рисунке 8-13**. Нажмите **ОК**.

Рисунок 8-13. Меню создания нового пользователя



Регистрация исходного кода

Последним шагом к подготовке компьютера для релиза с помощью **Maven** является регистрация проекта **gswm**, находящегося в папке <C:\apress\gswm-book\chapter8\gswm>, в новом репозитории. Используя командную строку переместитесь в папку <C:\apress\gswm-book\chapter8\gswm> и выполните следующую последовательность команд:

```
svn checkout http://localhost/svn/gswm/trunk/ C:/apress/gswm-book/chapter8/gswm --username gswm --password gswm
svn add src
svn add pom.xml
svn commit -m "Initial commit"
```

Результат выполнения этих команд отображен на **Рисунке 8-14**.

Рисунок 8-14. Результат начального коммита SVN

```
Administrator: Windows Command Processor
C:\apress\gswm-book\chapter8\gswm>svn checkout http://localhost/svn/gswm/trunk/
C:/apress/gswm-book/chapter8/gswm --username gswm --password gswm
Checked out revision 1.

C:\apress\gswm-book\chapter8\gswm>svn add src
A          src
A          src\test
A          src\test\java
A          src\test\java\HelloWorldTest.java
A          src\main
A          src\main\java
A          src\main\java\HelloWorld.java

C:\apress\gswm-book\chapter8\gswm>svn add pom.xml
A          pom.xml

C:\apress\gswm-book\chapter8\gswm>svn commit -m "Initial commit"
Adding          pom.xml
Adding          src
Adding          src\main
Adding          src\main\java
Adding          src\main\java\HelloWorld.java
Adding          src\test
Adding          src\test\java
Adding          src\test\java\HelloWorldTest.java
Transmitting file data ...
Committed revision 2.
```

Используя браузер перейдите на страницу <http://localhost/svn/gswm/trunk>. В ответ на вопрос введите имя пользователя **gswm** и пароль **gswm** и вы увидите зарегистрированный код. На **Рисунке 8-15** изображен вид экрана браузера.

Рисунок 8-15. Проект, зарегистрированный в SVN

Релиз с помощью Maven

Релиз артефакта с использованием процесса релиза **Maven** требует использования двух важных целей: **prepare** и **perform**. Дополнительно, плагин релиза предоставляет цель **clean**, которая пригодится, если что-то пойдет не так.

Цель Prepare

Цель **prepare** (подготовить), как и подразумевается из имени, подготавливает проект для релиза. В частности, на этой стадии **Maven** выполняет следующие операции:

- **check-poms**: проверяется, содержит ли номер версии, указанный в файле [pom.xml](#), слово **SNAPSHOT**;
- **scm-check-modifications**: проверяется, существуют ли изменения, не вошедшие в коммит;
- **check-dependency-snapshots**: проверяется, есть ли в файле [pom.xml](#) **SNAPSHOT**-зависимости. Наилучшим подходом является использование **RELEASE**-зависимостей. Любые **SNAPSHOT**-зависимости, найденные в файле [pom.xml](#), приведут к неудаче релиза;
- **map-release-versions**: если **prepare** запущен в интерактивном режиме, то у пользователя запрашивается номер релиза;
- **map-development-versions**: если **prepare** запущен в интерактивном режиме, то у пользователя запрашивается следующая версия разработки;
- **generate-release-poms**: генерирует файл [pom.xml](#) релиза;
- **scm-commit-release**: выполняет коммит релиза файла [pom.xml](#) в **SCM**;
- **scm-tag**: создает тег релиза для кода в **SCM**;
- **rewrite-poms-for-development**: обновляет файл [pom.xml](#) для следующего цикла разработки;
- **remove-release-poms**: удаляет файл [pom.xml](#), сгенерированный для релиза;
- **scm-commit-development**: отправляет файл [pom.xml](#) с версией разработки;
- **end-release**: завершает фазу **prepare** релиза.

Можно упростить задачу, предоставив **SCM** информацию в файле [pom.xml](#) проекта. **Листинг 8-3** содержит отрывок файла [pom.xml](#), содержащий такую информацию для **SCM**.

Listing 8-3. Файл `pom.xml` с информацией для **SCM**

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <!-- Content removed for brevity -->
  <scm>
    <connection>scm:svn:http://localhost/svn/gswm/trunk</connection>
    <developerConnection>scm:svn:http://localhost/svn/gswm/trunk</developerConnection>
    <url>http://localhost/svn/gswm/trunk</url>
  </scm>
  <!-- Content removed for brevity -->
</project>
```

После того, как вы обновили файл `pom.xml` на своем локальном компьютере, закоммитьте модифицированный файл в **SVN** путем выполнения следующей команды:

```
svn commit -m "Added SVN Information"
```

Вывод после выполнения этой команды отображен на **Рисунке 8-16**.

Рисунок 8-16. Вывод выполнения команды `svn commit`

```
C:\apress\gswm-book\chapter8\gswm>svn commit -m "Added SVN Information"
Sending          pom.xml
Transmitting file data .
Committed revision 3.
```

Для успешного взаимодействия **Maven** с **SVN**-сервером требуются соответствующие полномочия с правами записи на сервере. Вы предоставляете эту информацию в файле `settings.xml`, как отображено в **Листинге 8-4**. Значение **ID** для тега `<server>` объявлено как `localhost` должно соответствовать имени хоста **SVN**.

Листинг 8-4. Файл `pom.xml` с настройками **SVN**-сервера

```
<?xml version="1.0" encoding="UTF-8"?>
  <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>nexusReleases</id>
        <username>deployment</username>
```

```

    <password>deployment123</password>
  </server>
  <server>
    <id>nexusSnapshots</id>
    <username>deployment</username>
    <password>deployment123</password>
  </server>
  <server>
    <id>localhost</id>
    <username>gswm</username>
    <password>gswm</password>
  </server>
</servers>
</settings>

```

Теперь у вас есть вся информация, требуемая для цели **prepare Maven**. **Листинг 8-5** отображает результат выполнения цели **prepare**. Так как цель **prepare** была запущена в интерактивном режиме, то **Maven** будет запрашивать у вас версию релиза, тег или ярлык релиза и новую версию разработки. Принять предлагаемые **Maven** значения по умолчанию можно, нажимая **Enter** в ответ на каждый вопрос.

Листинг 8-5. Команда Maven release:prepare

```

C:\apress\gswm-book\chapter8\gswm>mvn release:prepare
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-release-plugin:2.3.2:prepare (default-cli) @ gswm ---
[INFO] Verifying that there are no local modifications...
[INFO] ignoring changes on: **\release.properties, **\pom.xml.next, **\
pom.xml.releaseBackup, **\pom.xml.backup, **\pom.xml.branch, **\pom.xml.tag
[INFO] Executing: cmd.exe /X /C "svn --username gswm --password *****
--noauth-cache --non-interactive status"
[INFO] Working directory: C:\apress\gswm-book\chapter8\gswm
[INFO] Checking dependencies and plugins for snapshots ...
What is the release version for "Getting Started with Maven"? (com.apress.gswmbook:gswm)
1.0.0: :
What is SCM release tag or label for "Getting Started with Maven"? (com.apress.gswmbook:gswm)
gswm-1.0.0: :
What is the new development version for "Getting Started with Maven"?
(com.apress.gswmbook:gswm) 1.0.1-SNAPSHOT: :
[INFO] Transforming 'Getting Started with Maven'...
-----

```

```

[INFO] [INFO] Building jar: C:\apress\gswm-book\chapter8\gswm\target\gswm-1.0.0.jar
[INFO] [INFO] -----
[INFO] [INFO] BUILD SUCCESS
[INFO] [INFO] -----
[INFO] [INFO] Total time: 1.654 s
[INFO] [INFO] Finished at: 2014-10-22T23:10:44-06:00
[INFO] [INFO] Final Memory: 11M/27M
[INFO] [INFO] -----
[INFO] Checking in modified POMs...
[INFO] Executing: cmd.exe /X /C "svn --username gswm --password *****
--noauth-cache --non-interactive commit --file C:\Users\<<USER_NAME>>\AppData\
Local\Temp\maven-scm-203076178.commit --targets C:\Users\<<USER_NAME>>\
AppData\Local\Temp\maven-scm-5496549062663519106-targets"
[INFO] Working directory: C:\apress\gswm-book\chapter8\gswm
[INFO] Tagging release with the label gswm-1.0.0...
[INFO] Executing: cmd.exe /X /C "svn --username gswm --password *****
--noauth-cache --non-interactive copy --file C:\Users\<<USER_NAME>>\AppData\
Local\Temp\maven-scm-85876759.commit --revision 6 http://localhost/svn/gswm/
trunk http://localhost/svn/gswm/tags/gswm-1.0.0"
[INFO] Working directory: C:\apress\gswm-book\chapter8\gswm
[INFO] Transforming 'Getting Started with Maven'...
[INFO] Not removing release POMs
[INFO] Checking in modified POMs...
[INFO] Executing: cmd.exe /X /C "svn --username gswm --password ***** --no-authcache
--non-interactive commit --file C:\Users\<<USER_NAME>>\AppData\Local\
Temp\maven-scm-112170711.commit --targets C:\Users\<<USER_NAME>>\AppData\
Local\Temp\maven-scm-244
0605286339680080-targets"
[INFO] Working directory: C:\apress\gswm-book\chapter8\gswm
[INFO] Release preparation complete.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 33.711 s
[INFO] Finished at: 2014-10-22T23:10:44-06:00
[INFO] Final Memory: 7M/17M
[INFO] -----

```

Обратите внимание, что команды **svn** исполняются как часть цели **prepare**. Успешное завершение цели **prepare** приведет к созданию тега **SVN**, как отображено на **Рисунке 8-17**. Файл **pom.xml** проекта **gswm** теперь имеет версию **1.0.1-SNAPSHOT**.

Рисунок 8-17. Тег **SVN**, созданный при выполнении цели **prepare**



Цель Clean

Цель **prepare** в процессе своего выполнения выполняет большое количество действий и генерирует временные файлы, такие, как [release.properties](#) и [pom.xml.releaseBackup](#). При успешном завершении цель **prepare** эти временные файлы удалит. Но если выполнение цели **prepare** не завершилось успешно (например, в случае неудачи связи с **SVN**-сервером), то проект остается в так называемом **грязном** состоянии. Здесь приходит на помощь цель **release:clean** (очистить) плагина релиза **Maven**. Как и предполагает её имя, эта цель удаляет все временные файлы, сгенерированные в процессе выполнения релиза.

Замечание: Цель **clean** должна использоваться только в случае неуспешного выполнения цели **prepare**.

Цель Perform

Цель **perform** отвечает за проверку кода из заново созданного тега, а также сборку и разворачивание кода релиза в удаленный репозиторий. Как часть цели **perform** выполняются следующие фазы:

- **verify-completed-prepare-phases**: проверяется, что перед целью **perform** была выполнена цель **prepare**;
- **checkout-project-from-scm**: проверяется код релиза из тега **SCM**;
- **run-perform-goal**: выполняет цели, ассоциированные с целью **perform**. По умолчанию это цель **deploy**.

Консольный вывод выполнения цели **perform** для проекта **gswm** отображена на **Листинге 8-6**.

Листинг 8-6. Команда Maven release:perform

```

C:\apress\gswm-book\chapter8\gswm>mvn release:perform
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Getting Started with Maven 1.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-release-plugin:2.3.2:perform (default-cli) @ gswm ---
[INFO] Checking out the project to perform the release ...
[INFO] Executing: cmd.exe /X /C "svn --username gswm --password *****
--noauth-cache --non-interactive checkout http://localhost/svn/gswm/tags/gswm-1.0.0
C:\apress\gswm-book\chapter8\gswm\target\checkout"
[INFO] Working directory: C:\apress\gswm-book\chapter8\gswm\target
-----
[INFO] [INFO] Installing C:\apress\gswm-book\chapter8\gswm\target\checkout\
target\gswm-1.0.0.jar to C:\Users\<<USER_NAME>>\.m2\repository\com\apress\
gswmbook\gswm\1.0.0\gswm-1.0.0.jar
[INFO] [INFO] Installing C:\apress\gswm-book\chapter8\gswm\target\checkout\
pom.xml to C:\Users\<<USER_NAME>>\.m2\repository\com\apress\gswmbook\
gswm\1.0.0\gswm-1.0.0.pom
[INFO] [INFO] Installing C:\apress\gswm-book\chapter8\gswm\target\checkout\
target\gswm-1.0.0-sources.jar to C:\Users\<<USER_NAME>>\.m2\repository\com\
apress\gswmbook\gswm\1.0.0\gswm-1.0.0-sources.jar
[INFO] [INFO] Installing C:\apress\gswm-book\chapter8\gswm\target\checkout\
target\gswm-1.0.0-javadoc.jar to C:\Users\<<USER_NAME>>\.m2\repository\com\
apress\gswmbook\gswm\1.0.0\gswm-1.0.0-javadoc.jar
[INFO] [INFO]
[INFO] [INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ gswm ---
[INFO] Uploading: http://localhost:8081/nexus/content/repositories/releases/
com/apress/gswmbook/gswm/1.0.0/gswm-1.0.0.jar
[INFO] 2/3 KB
[INFO] 3/3 KB
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/
com/apress/gswmbook/gswm/1.0.0/gswm-1.0.0.jar (3 KB at 13.4 KB/sec)
[INFO] Uploading: http://localhost:8081/nexus/content/repositories/releases/
com/apress/gswmbook/gswm/1.0.0/gswm-1.0.0.pom
[INFO] 2/3 KB
[INFO] 3/3 KB
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/
com/apress/gswmbook/gswm/1.0.0/gswm-1.0.0.pom (3 KB at 14.5 KB/sec)
[INFO] Downloading: http://localhost:8081/nexus/content/repositories/
releases/com/apress/gswmbook/gswm/maven-metadata.xml
[INFO]
[INFO] Uploaded: http://localhost:8081/nexus/content/repositories/releases/
com/apress/gswmbook/gswm/1.0.0/gswm-1.0.0-javadoc.jar (35 KB at 368.5 KB/sec)
[INFO] [INFO] BUILD SUCCESS

```

```

[INFO] [INFO] -----
[INFO] [INFO] Total time: 3.807 s
[INFO] [INFO] Finished at: 2014-10-22T23:26:36-06:00
[INFO] [INFO] Final Memory: 17M/42M
[INFO] [INFO] -----
[INFO] [INFO] Cleaning up after release...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.867 s
[INFO] Finished at: 2014-10-22T23:26:36-06:00
[INFO] Final Memory: 8M/19M
[INFO] -----

```

На этом релиз версии **1.0.0** проекта **gswm** завершен. Артефакт оказался в менеджере репозитория **Nexus**, как изображено на **Рисунке 8-18**.

Рисунок 8-18. Nexus с артефактом релиза

Index of /repositories/releases/com/apress/gswmbook/gswm/1.0.0

Name	Last Modified	Size	Description
Parent Directory			
gswm-1.0.0-javadoc.jar	Tue Oct 14 18:22:41 MDT 2014	35456	
gswm-1.0.0-javadoc.jar.md5	Tue Oct 14 18:22:41 MDT 2014	32	
gswm-1.0.0-javadoc.jar.sha1	Tue Oct 14 18:22:41 MDT 2014	40	
gswm-1.0.0-sources.jar	Tue Oct 14 18:22:40 MDT 2014	559	
gswm-1.0.0-sources.jar.md5	Tue Oct 14 18:22:40 MDT 2014	32	
gswm-1.0.0-sources.jar.sha1	Tue Oct 14 18:22:40 MDT 2014	40	
gswm-1.0.0.jar	Tue Oct 14 18:22:40 MDT 2014	2362	
gswm-1.0.0.jar.md5	Tue Oct 14 18:22:40 MDT 2014	32	
gswm-1.0.0.jar.sha1	Tue Oct 14 18:22:40 MDT 2014	40	
gswm-1.0.0.pom	Tue Oct 14 18:22:40 MDT 2014	2129	
gswm-1.0.0.pom.md5	Tue Oct 14 18:22:40 MDT 2014	32	
gswm-1.0.0.pom.sha1	Tue Oct 14 18:22:40 MDT 2014	40	

Итоги

Менеджеры внутренних репозиториев, такие, как Nexus, позволяют предприятиям в полной мере применять **Maven**. Кроме выполнения роли прокси для публичных репозиториев, они позволяют обмениваться и управлять компонентами. В этой главе была рассмотрена интеграция **Maven** с **Nexus**, а также пройден по шагам процесс разворачивания артефакта в **Nexus**. Кроме того, вы ознакомились с процессом релиза **Maven** и его отдельными фазами.

На этом наше путешествие завершается. Из этой книги вы узнали ключевые концепции, лежащий в основе **Maven**. Мы надеемся, что вы воспользуетесь полученными о **Maven** знаниями для автоматизации и улучшения существующих у вас процессов управления проектами.

Предметный указатель

A

Apache Ant, 9
Apache Archiva, 21
Apache Ivy, 9
Apache Jakarta Alexandria, 6
Apache Maven, 6
Apache Subversion (SVN), 17
Apache Tile, 8
Apache Turbine, 6
Artifactory, 21

B

Bamboo, 7

C

CSS, 48

D

Domain Specific Language (DSL), 10

E

EAR, 23
Eclipse, 6
Enterprise JavaBean (EJB), 52

G

GAV-координаты, 23
Gradle, 10
Groovy, 10

H

hamcrest, 34
Hudson, 7

J

James Server, 28
JAR, 23
JAR-файл, 19
Java Development Kit (JDK), 12
Java Enterprise Edition (JEE), 52
javac, 9
Jenkins, 7
JUnit, 24
JUnit.jar, 34

L

Log4J, 19

M

Maven 1.0, 6
maven-archetype-mojo, 45

N

NetBeans, 6
Nexus, 77

R

Ruby on Rails, 8

S

SCM, 27
 Git, 27
 SVN, 27
SiteMesh, 8
Sonatype, 8
Sonatype Nexus, 21
Spring, 28
Subversion (SVN), 81

SVN, 19

T

Tomcat, 28

V

Velocity, 28

W

WAR, 9, 23

Windows 7, 12

A

аннотация @Mojo, 45

архетип

- maven-archetype-webapp, 49

архетипы Maven, 8, 48

архитектура, основанная на плагинах, 7

Б

бинарный zip-файл Maven 3.2.3, 12

В

версии JDK для Maven, 12

встроенный веб-сервер

- Jetty, 50

- Tomcat, 50

Д

декларативное управление зависимостями, 19

директории Maven, 28

добавление JUnit-теста, 33

добавление логотипа сайта, 70

Ж

жизненный цикл

- Clean, 42

- Default, 42

- Site, 42

жизненный цикл сборки, 42

З

зависимость

- maven-plugin-api, 45

задача

- Ant, 9

И

инструмент

- Cobertura, 74

- Emma, 74

- FindBugs, 75

инструмент командной строки, 7

интеграционные тесты, 28

К

квалификатор

- SNAPSHOT, 30

клиентский JAR, 52

код HTTP-состояния «200», 56

команда

- clean:clean, 41

- mvn archetype:generate, 61

- mvn compiler:compile, 42

- mvn dependency:tree, 34

- mvn package, 31, 56

- mvn site, 64

- mvn tomcat7:run, 62

- помощь Maven, 15

команды

- версия Maven, 15

- параметры установки Maven, 15

конфликты версий Maven, 24

М

менеджер хранилища, 21

Н

настройка прокси, 18

О

область видимости

compile, 25

import, 25

provided, 25

runtime, 25

system, 25

test, 25

объем памяти JVM, 14

окно свойств системы, 13

открытый проект, 8

ошибка

«Unable to download artifact», 18

П

папка

.m2, 16

apache-maven-3.2.3-bin, 12

apidocs, 72

apt, 69

c:\tools\maven, 12

cobertura, 75

lib, 19

surefire-reports, 74

WebContent, 6

WebPages, 7

папка репозитория, 16

параметр

interactiveMode, 54

package, 54

переменная окружения

M2_HOME, 13

переменная окружения

Path, 14

переменная окружения

MAVEN_OPTS, 14

плагин

install, 25

Javadoc, 72

Surefire, 73

плагины Maven, 40

поддержка IDE, 7

поддержка Maven IDE, 18

Покрывтие кода, 74

Р

разработка целей Maven, 44

репозиторий

Releases, 79

Snapshots, 79

С

сайт Apache Maven, 12

свойство

maven.test.skip, 44

секция

<plug-in>, 41

<reporting>, 73

сервер VisualSVN, 82

синтаксис запуска цели Maven, 47

синтаксическая конструкция $\${...}$, 37

соглашение об обозначении версий, 30

соглашение по конфигурации (CoC), 8, 44

стандартизированная структура папок, 6

Т

тег @goal, 45

тип упаковки

maven-plugin, 45

транзитивные зависимости, 23

У

унифицированный интерфейс, 73

установка

создание системной переменной, 14

Ф

фаза

compile, 43
 deploy, 43
 install, 43
 package, 43
 test, 43
 validate, 43
 фазы Maven, 42
 файл
 build.gradle, 10
 build.xml, 9
 findbugs.html, 76
 index.apr, 69
 ivy.xml, 9
 pom.xml, 7, 19
 settings.xml, 16, 79
 site.xml, 69
 формат Almost Plain Text (APT), 69
 фреймворк Spring, 8

Х

хранилище
 JBoss, 22
 Spring, 22

Ц

цель
 Ant, 9
 compile, 9, 40
 create, 69
 generate, 48
 perform, 93
 prepare, 89

Централ Maven (Maven Central), 19

Э

элемент

<description>, 67
 <distributionManagement>, 79
 <exclusions>, 38
 <finalName>, 42
 <license>, 67
 <mailingList>, 67
 <menu>, 70
 <modules>, 55
 <packaging>, 44
 <properties>, 37
 <reporting>, 72
 <repository>, 79
 <snapshotRepository>, 79

элемент settings.xml

activeProfile, 17
 interactiveMode, 17
 localRepository, 17
 mirrors, 17
 offline, 17
 profiles, 17
 proxies, 17
 servers, 17

Ю

Юнит-тесты, 73

Я

язык DSL, 10

Introducing Maven

Balaji Varanasi

Sudha Belida

Apress®

Introducing Maven

Copyright © 2014 by Balaji Varanasi and Sudha Belida

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

ISBN-13 (pbk): 978-1-4842-0842-7

ISBN-13 (electronic): 978-1-4842-0841-0

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr

Lead Editor: Steve Anglin

Technical Reviewer: Deepak Vohra

Developmental Editor: Gary Schwartz

Editorial Board: Steve Anglin, Mark Beckner, Gary Cornell, Louise Corrigan, Jonathan Gennick,

Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson,

Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing, Matt Wade

Coordinating Editor: Mark Powers

Copy Editor: Mary Bearden

Compositor: SPi Global

Indexer: SPi Global

Artist: SPi Global

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com , or visit www.springeronline.com . Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com , or visit www.apress.com .

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/bulk-sales .

Any source code or other supplementary material referenced by the author in this text is available to readers at www.apress.com/9781484208427 . For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Посвящается нашим родителям

Содержание

Введение.....	4
Глава 1: Начало работы с Maven.....	6
Стандартная структура папок.....	6
Декларативное управление зависимостями.....	7
Плагины.....	7
Единая абстракция сборки.....	7
Поддержка инструментов.....	7
Архетипы.....	8
Открытый исходный код.....	8
Альтернативы Maven.....	9
Ant + Ivy.....	9
Gradle.....	10
Итоги.....	11
Глава 2: Установка Maven.....	12
Проверка установки.....	15
Получение помощи.....	15
Дополнительные настройки.....	16
Настройка прокси.....	18
Поддержка IDE.....	18
Итоги.....	18
Глава 3: Управление зависимостями Maven.....	19
Использование хранилищ.....	22
Идентификация зависимости.....	23
Транзитивные зависимости.....	23
Область видимости зависимости.....	24
Ручная установка зависимостей.....	25
Итоги.....	26
Глава 4: Основы Maven-проекта.....	27
Организация основы проекта.....	27
Содержимое файла pom.xml.....	29
Сборка проекта.....	31
Тестирование проекта.....	33
Свойства в pom.xml.....	37
Исключение зависимостей.....	38
Итоги.....	39

Глава 5: Жизненный цикл Maven	40
Цели и плагины	40
Жизненный цикл и фазы	42
Разработка плагинов	44
Итоги	47
Глава 6: Архетипы Maven	48
Встроенные архетипы.....	48
Создание Веб-проекта	49
Мультимодульный проект	52
Создание архетипов	56
Использование архетипов.....	61
Итоги	63
Глава 7: Документация и отчетность.....	64
Использование жизненного цикла сайта	64
Дополнительное конфигурирование сайта.....	69
Генерация отчетов Javadoc.....	72
Генерация отчетов юнит-тестирования	73
Генерация отчетов о покрытии кода.....	74
Генерация отчета FindBugs.....	75
Итоги	76
Глава 8: Публикация с помощью Maven	77
Интеграция с Nexus.....	77
Релиз проекта.....	81
Установка инструмента командной строки Subversion	82
Создание репозитория	85
Регистрация исходного кода.....	87
Релиз с помощью Maven.....	89
Цель Prepare	89
Цель Clean.....	93
Цель Perform.....	93
Итоги	96
Предметный указатель.....	97
Introducing Maven	101
Об авторах	107
О техническом редакторе	108
Благодарности.....	109

Об авторах



Balaji Varanasi является менеджером по разработке программного обеспечения, автором, лектором и технологическим предпринимателем. Он обладает более, чем 14-летним опытом проектирования и разработки высокопроизводительных и масштабируемых мобильных приложений Java и .Net. За этот период он поработал в областях безопасности, веб-доступа, поисковых и корпоративных приложений. Обладая степенью магистра информатики Государственного Университета Юты он работает помощником на факультете Университета Феникса, преподавая на курсах программирования и информационных систем. Он делится своими мыслями и экспериментами на <http://blog.inflinx.com>.



Sudha Belida является ведущим разработчиком и энтузиастом технологий. Она обладает более чем семилетним опытом работы с Java и JEE-технологиями и фреймворками, такими, как Spring, Hibernate, Struts и AngularJS. Её интересы лежат в предпринимательстве и гибких методологиях проектирования и разработки программного обеспечения. Она обладает степенью магистра в области вычислений Университета Юты. В свободное время она обожает путешествовать и наслаждаться природой штата Юта.

О техническом редакторе



Деерак Vohra является консультантом и ведущим участником компании-разработчика программного обеспечения NuBean.com. Являясь сертифицированным программистом Java и разработчиком веб-компонентов, он имеет более чем 5-летний опыт работы в таких областях, как XML, Java-программирование и JEE-технологии. Деерак является соавтором книги *Pro XML Development with Java Technology* (Apress, 2006), а также автором таких книг, как *JDBC 4.0 and Oracle JDeveloper for J2EE Development*, *Processing XML Documents with Oracle JDeveloper 11g*, *EJB 3.0 Database Persistence with Oracle Fusion Middleware 11g*, и *Java EE Development in Eclipse IDE* (Packt Publishing). Также он выступает техническим редактором книг *WebLogic: The Definitive Guide* (O'Reilly Media, 2004) и *Ruby Programming for the Absolute Beginner* (Cengage Learning PTR, 2007).

Благодарности

Появление этой книги было бы невозможным без поддержки нескольких человек, и мы пользуемся данной возможностью искренне их поблагодарить.

Спасибо парням из Apress: Steve Anglin, Mark Powers, Matthew Moodie и многим другим. Мы также выражаем огромную благодарность Деерак Vohra за его технические правки и за ценнейшие отзывы.

И наконец, мы хотели бы поблагодарить наших родителей за их постоянную поддержку и ободрение. Без них появление этой книги было бы невозможным.