序

誠摯感謝 立法委員暨前金管會主委 曾銘宗先生、宏達電 謝昱帆副總經理、華碩電腦 郭海威處長等對本書的肯定與推薦！繼：

1. Java SE8 OCAJP 專業認證指南
2. Java SE8 OCPJP 進階認證指南
3. Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap

後，個人的第 4 本著作《Spring Boot 情境式網站開發指南：使用 Spring Data JPA、Spring Security、Spring Web Flow》終於完成了！

本書介紹 4 個 Spring 框架模組，不少內容是前著作《Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap》的延伸：

1. Spring Data JPA
   這部分需要 ORM 與 JPA 的基礎，也是一個大主題，因此使用了一半的篇幅說明，也是前著作章節「15.3 Object Relational Mapping (ORM) 的軟體架構與 JPA」的接續。

2. Spring Security
   這部分需要資訊安全的完整概念。融合前著作章節「17 網站安全性實作」與本書對 OWASP TOP 10 的整理，可以讓讀者在閱讀本篇章時更能體會 Spring Security 不凡之處。

3. Spring Boot
   Spring Boot 是 Spring 4 之後的重大突破，簡化了 Spring 的開發，已經是目前使用 Spring 框架的基本配備。

4. Spring Web Flow
   這部分是本書的綜合實作。除了說明並驗證「情境式網站」概念如何提高使用者體驗外，希望對筆者曾任職的鐵路運輸公司的資訊部門所關注的主題能有幫助，也了卻一個未完成的責任。

限於篇幅與秉持的教學習慣，書中內容若涉及或傳承前書知識，將請讀者參閱指定篇章。

這本書的出版，受到了很多親朋好友、公司同事、讀者、臉書 Java 技術與認證交流平台 (https://www.facebook.com/groups/748837991920629/?ref=bookmarks) 版友的鼓勵與督促，再次感謝各位。

曾瑞君

謹致 2020.05

01

使用 Maven 管理 Java 專案

章節重點

1.1 認識 Maven 的專案管理思維與架構
1.2 使用 Maven 指令建立專案
1.3 結合 Maven 與 Eclipse

1.1 認識 Maven 的專案管理思維與架構

1.1.1 Maven 的安裝和設定

Maven 簡介

Maven 是 Apache 基金會推廣的一個軟體專案管理工具。它使用「Project Object Model, (POM)」檔案來管理專案的建構、關聯性和說明文件，強項在於可以自動下載和專案相關的函式庫 (libraries)。

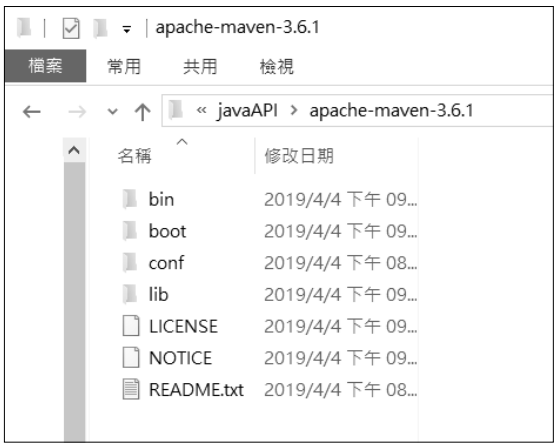
一般若要分享一個 Java 的專案給其他人的時候，最大的麻煩是必須連帶複製專案需要的 JAR 檔。若有多個 Java 專案的時候，必然發現相同的 JAR 檔必須複製多次放在不同的專案裡，因此浪費硬碟空間；另下載相同 JAR 檔也浪費網路頻寬與時間。

使用 Maven 可以解決上述傳統專案管理的不便之處。Maven 的管理設定主要依賴 pom.xml 進行，將在後續內容說明。

Maven 的安裝和設定

Step01 安裝 JDK (本書使用 JDK 1.8)，並設定 JAVA_HOME 環境變數。

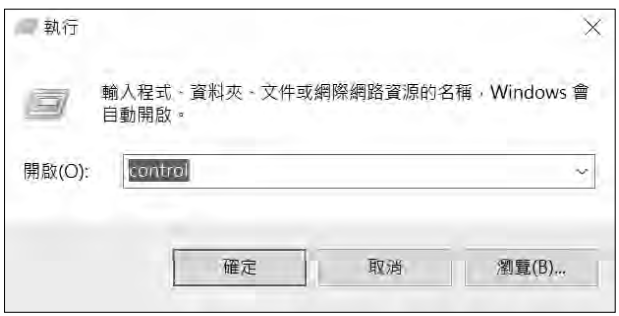
Step02 於「http://maven.apache.org/download.cgi」下載 Apache Maven，本書使用「apache-maven-3.6.1-bin.zip」。Maven 是免安裝的設計，只要下載壓縮檔再解壓縮即可使用。解壓縮後資料目錄內容如下：



▲ 圖 1-1 Maven 解壓縮後目錄

Step03 筆者將 Maven 解壓縮至路徑 D:\IDE\apache-maven-3.6.1，再建立環境變數「M2_HOME」、「MAVEN_HOME」指向該路徑。以 Window 10 為例，依以下圖例進行：

1. 開啟「執行」視窗：



▲ 圖 1-2 於「執行」視窗輸入「control」以開啟控制台

2. 開啟「控制台」視窗，並點擊「系統」圖示：

▲ 圖 1-3 控制台視窗

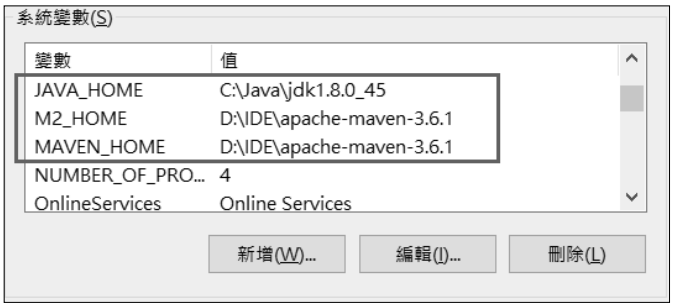
3. 開啟「系統」視窗，並點擊「進階系統設定」連結：

▲ 圖 1-4 系統視窗

4. 點擊頁籤「進階」，並點擊「環境變數」：

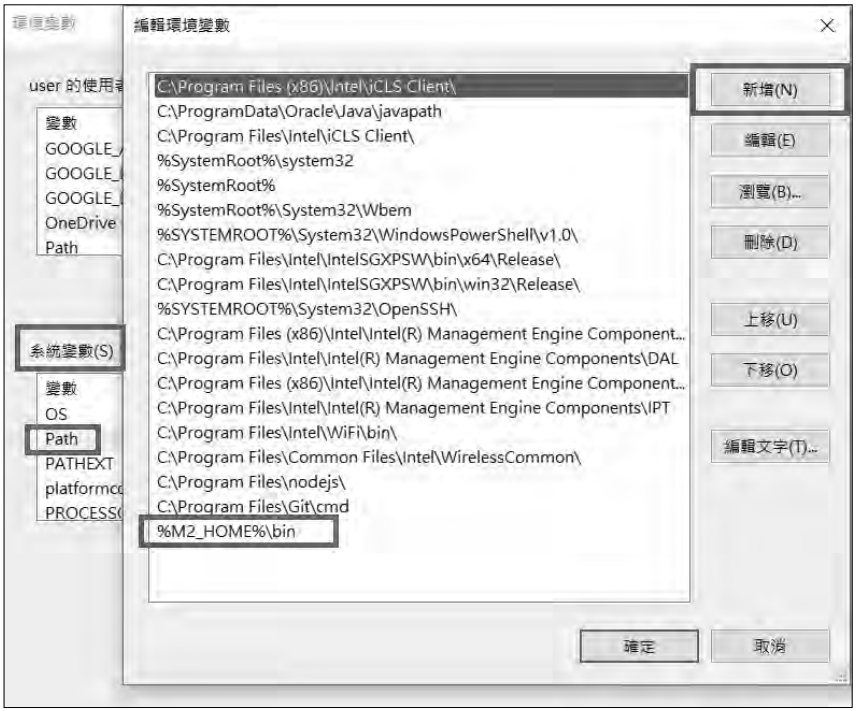
▲ 圖 1-5 系統進階設定

5. 設定環境變數 JAVA_HOME、M2_HOME、MAVEN_HOME：



▲ 圖 1-6 環境變數設定

Step04 修改 Path 環境變數，在尾端加上「%M2_HOME%\bin」，讓 Maven 的指令可以在任何路徑下執行：



▲ 圖 1-7 修改 Path 環境變數

Step05 使用 Maven 指令「mvn -version」驗證設定是否完成，可得到 Maven 的版本號：

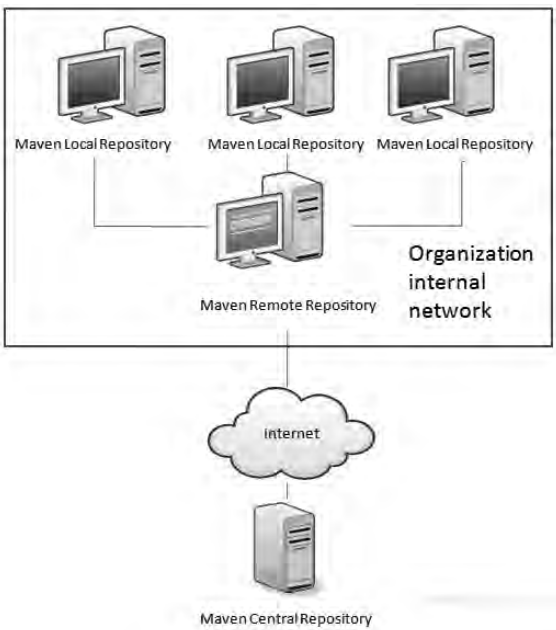
▲ 圖 1-8 指令 mvn -version 執行結果

1.1.2 Maven 儲存庫 (Repository)

當 Java 的專案管理使用 Maven 時，專案使用的 JAR 檔不會儲存在各自專案裡，而是統一放置在 Maven 儲存庫 (repository)，也就是 Maven 用來儲存 JAR 檔的地方，因此不同專案可以共用同一個 JAR 檔。Maven 儲存庫有 3 種：

1. local
2. central
3. remote

示意圖如下：



▲ 圖 1-9 Maven 儲存庫分類 (參考自 www.java2s.com)

Local 儲存庫

local 儲存庫是本地端 (local) 電腦用來儲存 Maven 的專案相關下載檔案 (通常是 JAR 檔) 的資料夾。當我們開始建構 (build) 一個 Maven 的 Java 專案時，該 Java 專案關聯的 JAR 就會被下載到 local 儲存庫，而非放置在專案內部；專案也會自動產生設定檔建立連結關係。所以不同專案可以共用同一 JAR 檔案，不需複製多次。預設位置依作業系統不同而異：

1. Unix/Linux： ~/.m2
2. Windows：C:\Users\ 使用者名稱 \.m2

可以藉由修改 Maven 設定檔重新指定 local 儲存庫：

1. 找到 xml 設定檔：{M2_HOME}\conf\setting.xml
2. 更新標籤「localRepository」值如下：

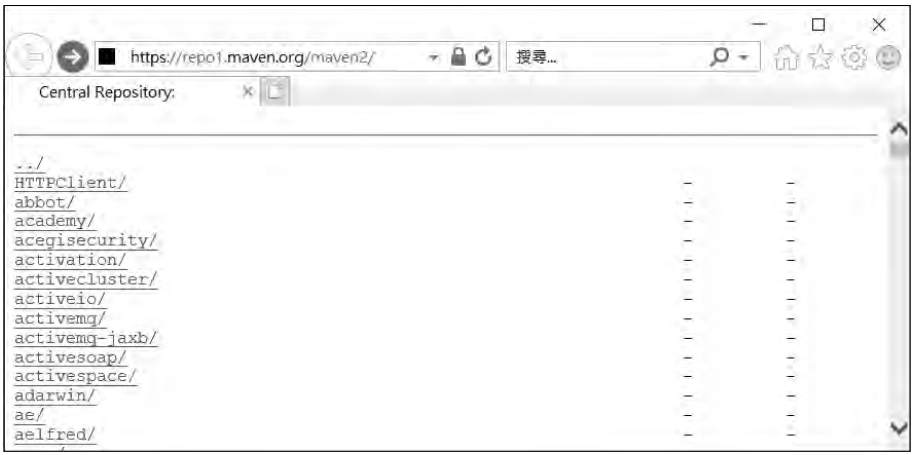
```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
          http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository                    預設
  <localRepository>/path/to/local/repo</localRepository>
  -->                                                        新增
  <localRepository>D:\IDE\maven_repo</localRepository>
```

▲ 圖 1-10 由 setting.xml 修改 local 儲存庫位置

Central 儲存庫

當開始建構 (build) 一個 Maven 專案時，Maven 會檢查個別專案的「pom.xml」的設定內容，決定下載哪些和專案相關的檔案。

預設 Maven 會檢查 local 儲存庫是否有需要的檔案。若沒有，則會由預設的 central 儲存庫，也就是「https://repo1.maven.org/maven2/」進行下載：



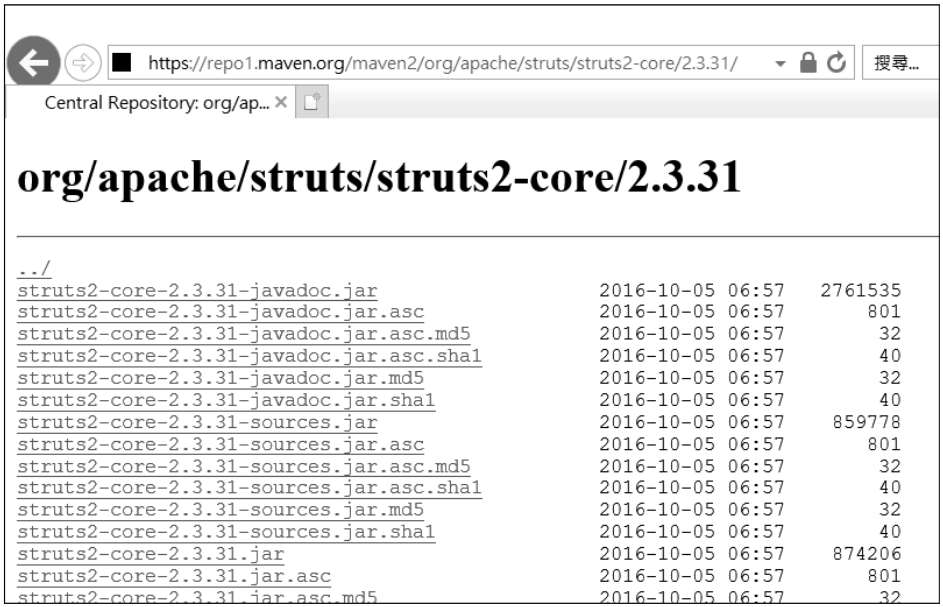
▲ 圖 1-11 central 儲存庫首頁

下載時根據 pom.xml 需要的關聯性，主要以標籤 <dependency> 敘述，下載專案需要的檔案。如以下範例將至「https://repo1.maven.org/maven2/」下載「struts2-core-2.3.31.jar」：

教學碼

```
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>2.3.31</version>
</dependency>
```

也就是到 https://repo1.maven.org/maven2/**org/apache/struts/struts2-core/2.3.31/** 下載需要的 struts2-core-2.3.31.jar 檔案：



▲ 圖 1-12 central 儲存庫下載頁面

可取得的檔案分 4 類：

1. javadoc
2. source
3. JAR
4. POM

其中的「*.pom」檔記錄了和該 JAR 檔有關聯的其他 JAR 檔，會視需要一併被下載，因此 Maven 可自動管理關聯 JAR 檔；若未指定 <version> 標籤，將會自動下載最新版。

Remote 儲存庫

Maven 預設會由 central 儲存庫下載需要的 JAR 檔，也就是「https://repo1.maven.org/maven2/」。但有些時候會需要由其他地方下載不足的檔案，常見如「java.net」和「JBoss repository」，可以在專案的 POM.xml 中使用標籤 <repository> 指定，範例如下：

教學碼

```
<project ...>
    <repositories>
        <repository>
            <id>java.net</id>
            <url>https://maven.java.net/content/repositories/public/</url>
        </repository>
        <repository>
            <id>JBoss repository</id>
            <url>https://repository.jboss.org/nexus/content/groups/public/</url>
        </repository>
    </repositories>
</project>
```

1.2 使用 Maven 指令建立專案

使用 Maven 可以建立不同種類的專案類型。使用以下方式：

語法

```
mvn archetype:generate > templates.txt
```

藉由導出的檔案，可以了解 Maven 支援的專案類型。

Maven 使用「模板」建立專案，和 Java 比較相關的有 2 種模板：

1. 建構 Java SE (JAR) 專案，使用「maven-archetype-quickstart」。
2. 建構 Java Web (WAR) 專案，使用「maven-archetype-webapp」。

使用 Maven 模板建構 Java 專案的基本指令為：

語法

```
mvn archetype:generate
-DgroupId = 專案使用的 package 根目錄 , 如 com.xxx
-DartifactId = 專案名稱
-DarchetypeArtifactId = 模板名稱
-DinteractiveMode = false
```

1.2.1 使用 maven-archetype-quickstart 模板建立 Java SE 專案

以在本書範例資料夾 c01\mvnProjects 內執行指令為例：

教學碼

```
mvn archetype:generate
-DgroupId=com.jim
-DartifactId=MyJavaTest
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

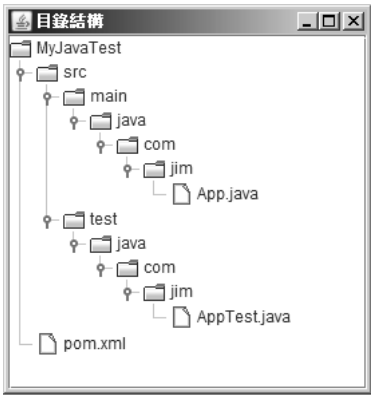
執行過程如下：

結果

```
F:\c01\mvnProjects>mvn archetype:generate -DgroupId=com.jim -DartifactId=MyJavaTest
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.1.2:generate (default-cli) > generate-sources @
standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.1.2:generate (default-cli) < generate-sources @
standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.1.2:generate (default-cli) @ standalone-pom
---
```

```
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: F:\c01\mvnProjects
[INFO] Parameter: package, Value: com.jim
[INFO] Parameter: groupId, Value: com.jim
[INFO] Parameter: artifactId, Value: MyJavaTest
[INFO] Parameter: packageName, Value: com.jim
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: F:\c01\mvnProjects\
MyJavaTest
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  40.658 s
[INFO] Finished at: 2019-10-31T22:42:02+08:00
[INFO] ------------------------------------------------------------------------
```

將在本書範例資料夾 c01\mvnProjects 內產生專案 MyJavaTest：



▲ 圖 1-13 Maven 專案 MyJavaTest 的目錄結構

特色是：

1. 路徑「src」下同時產生「main」和「test」資料夾。
   - main：撰寫程式碼的地方。
   - test：撰寫單元測試程式碼的地方。
2. 可以看到套件為 com.jim 的專案套件結構。
3. 產出的 pom.xml 是 Maven 的基本專案描述檔，可根據需求再修改：

範例：MyJavaTest\pom.xml

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
   org/maven-v4_0_0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <groupId>com.jim</groupId>
6      <artifactId>MyJavaTest</artifactId>
7      <packaging>jar</packaging>
8      <version>1.0-SNAPSHOT</version>
9      <name>MyJavaTest</name>
10     <url>http://maven.apache.org</url>
11     <dependencies>
12         <dependency>
13             <groupId>junit</groupId>
14             <artifactId>junit</artifactId>
15             <version>3.8.1</version>
16             <scope>test</scope>
17         </dependency>
18     </dependencies>
19 </project>
```

說明

5 6	使用標籤 <groupId> 與 <artifactId> 識別專案。標籤 <groupId> 相似於類別的 package 名稱，標籤 <artifactId> 相似於類別名稱。
8	使用標籤 <packaging> 表示 project 打包後的型態，可能為 JAR、WAR、EAR 或 POM，若是 android 則為 APK。
9-10	標籤 <name> 與 <url> 為非必要設定，用於輔助專案描述並協助識別。
11-18	使用標籤 <dependency> 設定專案關聯的函式庫，Java 專案主要是 JAR 檔案；並以群組標籤 <dependencies> 包含所有關聯函式庫。 標籤 <scope> 說明函式庫的用法： 表 1-1 標籤 <scope> 常用值

<scope> 值	說明
compile	此為預設值，表示該 JAR 檔用於協助編譯程式碼，且用於生命週期的所有階段 (phase)；部署時將一併被打包。
test	表示此 JAR 檔只有在測試時才需要使用，如 junit。部署時將不會一併打包。

	runtime	表示此 JAR 檔編譯時期不需要，只有在執行時才需要使用，如 JDBC driver。
	provided	該 JAR 檔預期由容等執行時期環境提供，如 servlet.jar、jsp-api.jar 等。部署時不會一併打包。
	system	跟 provided 相似，但非由程式執行時期環境如容器等提供，必須另外藉由 <systemPath> 標籤指定外部位置。
12-17	Maven 建構專案時自動建立 test 資料夾供撰寫單元測試程式碼使用，因此專案預設包含 junit 函式庫。	

1.2.2 使用 maven-archetype-webapp 模板建立 Java Web 專案

以在本書範例資料夾 c01\mvnProjects 執行指令為例：

教學碼

```
mvn archetype:generate
-DgroupId=com.jim
-DartifactId=MyJavaWebTest
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

執行過程如下：

結果

```
F:\c01\mvnProjects>mvn archetype:generate -DgroupId=com.jim
-DartifactId=MyJavaWebTest  -DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.1.2:generate (default-cli) > generate-sources @
standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.1.2:generate (default-cli) < generate-sources @
standalone-pom <<<
```

```
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.1.2:generate (default-cli) @ standalone-pom
---
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-webapp:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: F:\c01\mvnProjects
[INFO] Parameter: package, Value: com.jim
[INFO] Parameter: groupId, Value: com.jim
[INFO] Parameter: artifactId, Value: MyJavaWebTest
[INFO] Parameter: packageName, Value: com.jim
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: F:\c01\mvnProjects\
MyJavaWebTest
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  31.668 s
[INFO] Finished at: 2019-10-31T23:34:15+08:00
[INFO] ------------------------------------------------------------------------
```

將在本書範例資料夾 c01\mvnProjects 內產生專案 MyJavaWebTest：

▲ 圖 1-14 Maven 專案 MyJavaWebTest 的目錄結構

特色是：

1. 目錄 src 下產生 main 資料夾，開發程式碼時需要自己加上 package 根目錄。
2. 產出的「pom.xml」是 Maven 的基本專案描述檔，可根據需求再修改。與 Java SE 專案主要差別為行 7 的 <packaging> 標籤值為 war，打包專案時將產出 WAR 檔案：

範例：MyJavaWebTest\pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.jim</groupId>
    <artifactId>MyJavaWebTest</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>MyJavaWebTest 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>MyJavaWebTest</finalName>
    </build>
</project>
```

1.2.3 Maven 專案的生命週期

Maven 主要用來建構專案，所有流程和步驟在官網 https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html 都有清楚定義。在 Maven 專案的建構生命週期裡，每個階段 (phase) 都有預定要達成的目標，主要有：

表 1-2 Maven 專案階段目標

階段	說明
validate	驗正專案是否正確，必要資訊是否完備。
compile	編譯程式碼。
test	使用合適的框架進行單元測試 (unit testing)。
package	將編譯後的程式碼打包成可部署的型式，如 JAR 檔、WAR 檔。
verify	進行整合測試 (integration testing)，確保專案品質。

階段	說明
install	將本專案打包後儲存到 local 儲存庫，以分享給本機其他專案。
deploy	將本專案打包後儲存到 remote 儲存庫，以分享給其他專案。

各階段執行目標和 pom.xml 的設定內容息息相關。如在「package」階段將依據標籤 <packaging> 的內容決定打包後的型態，常見如 JAR 檔或 WAR 檔：

教學碼

```
<packaging>jar</packaging>
<packaging>war</packaging>
```

執行方式是在有 pom.xml 的路徑位置開啟 DOS 視窗並輸入階段名稱，以 package 為例：

教學碼

```
mvn package
```

就可以進行該階段工作任務，本例為打包。且先前的階段如「validate」、「compile」和「test」等都將被執行。

打包後的 JAR 檔或 WAR 檔將建立在專案內的「target」資料夾。使用以下指令將清理「target」資料夾內的檔案和 cache 資料：

教學碼

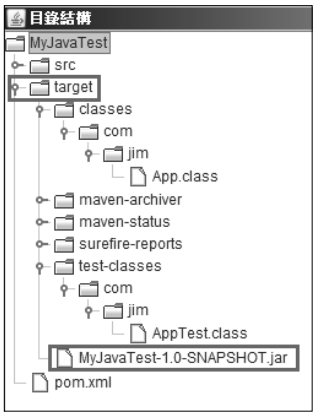
```
mvn clean
```

也可以合併使用以下指令，將確保打包的是最新的 JAR 或 WAR 檔案而沒有舊資料被 cache 的狀況：

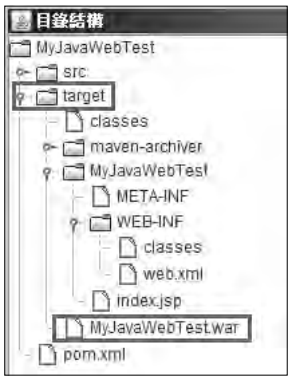
教學碼

```
mvn clean package
```

分別在上一節使用 Maven 指令建立的專案資料夾內，在與 pom.xml 檔案的同一層目錄執行指令「mvn package」，將打包專案後產出 MyJavaTest-1.0-SNAPSHOT.jar 與 MyJavaWebTest.war：



▲ 圖 1-15 打包 JAR 檔案



▲ 圖 1-16 打包 WAR 檔案

打包過程中反覆出現警告訊息：

結果

```
[WARNING] Using platform encoding (MS950 actually) to copy filtered resources,
i.e. build is platform dependent!
```

提示打包時使用的編碼語系為作業系統預設的 MS950，因此可以回到兩個專案的 pom.xml 分別指定改用 UTF-8 的編碼方式：

教學碼

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

以 MyJavaWebTest\pom.xml 為例，加上行 17-19：

範例：MyJavaWebTest\pom.xml

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
   org/maven-v4_0_0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <groupId>com.jim</groupId>
6      <artifactId>MyJavaWebTest</artifactId>
7      <packaging>war</packaging>
8      <version>1.0-SNAPSHOT</version>
9      <name>MyJavaWebTest Maven Webapp</name>
```

```
    <url>http://maven.apache.org</url>
    <dependencies>
        略…
    </dependencies>
    <build>
        <finalName>MyJavaWebTest</finalName>
    </build>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

並執行以下指令重新打包：

教學碼

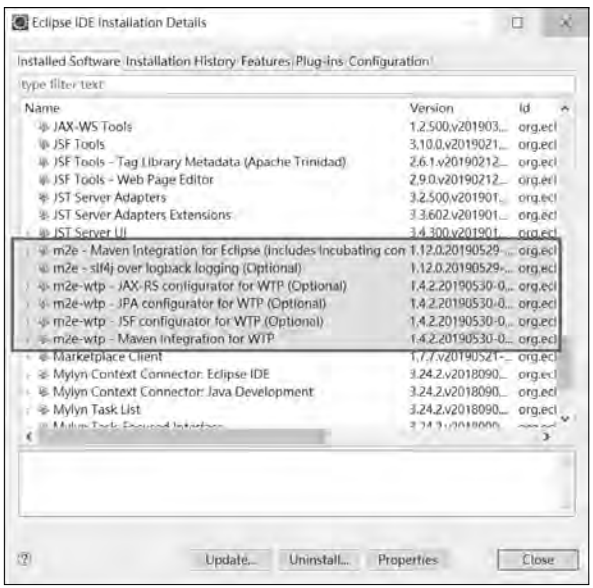
```
mvn clean package
```

選項 clean 先清除上一次打包檔案，將發現原本警告訊息已經消失不見。

1.3 結合 Maven 與 Eclipse

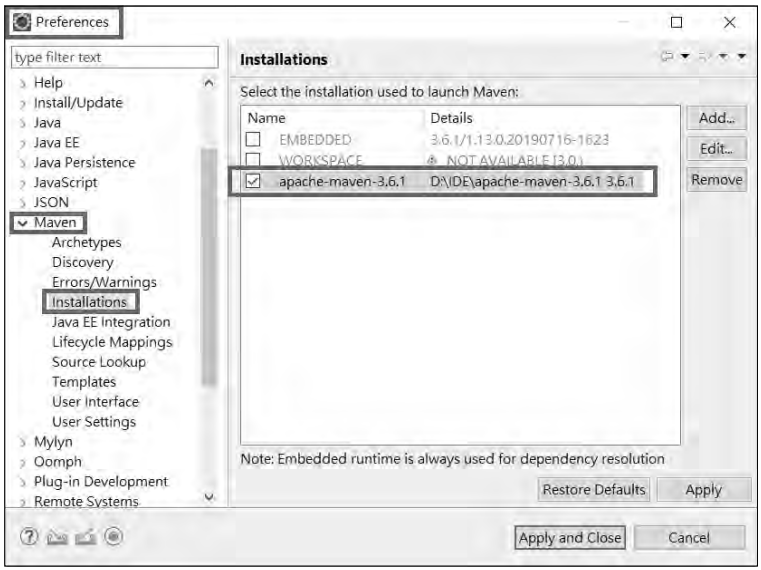
1.3.1 Eclipse 內建 Maven

過去 Eclipse 必須要自己安裝相關 plugin 才能使用 Maven，較新版的 Eclipse 已經改為內建，查詢 Eclipse 安裝的軟體項目可以找到「m2e」相關 plugin 如右：



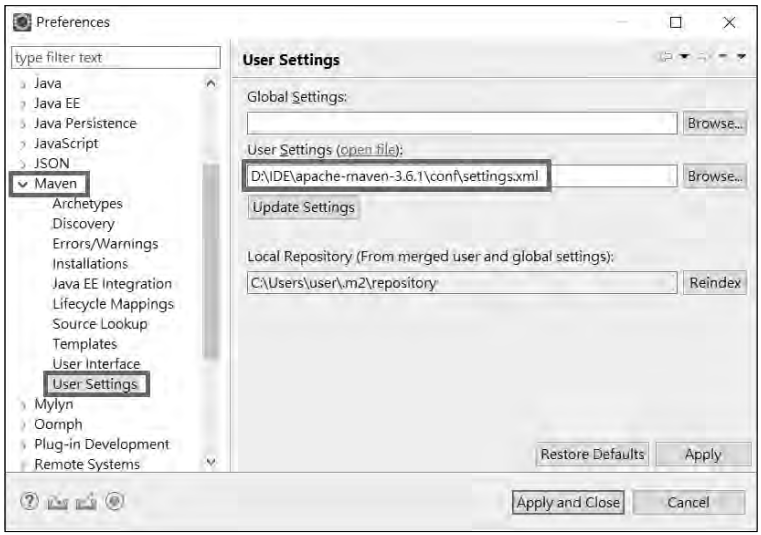
▲ 圖 1-17 Eclipse 預設安裝 Maven plugin

接下來開啟「Preferences」視窗，展開「Maven」節點並點選「Installations」，加入 Maven 安裝目錄「D:\IDE\apache-maven-3.6.1」的設定：



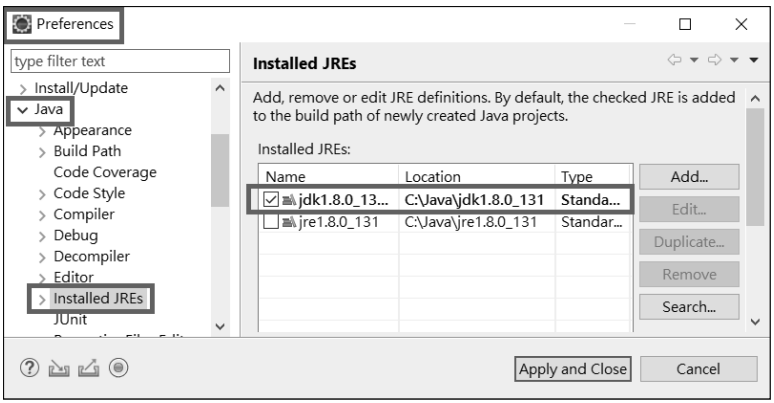
▲ 圖 1-18 設定 Maven 安裝路徑

展開「Maven」節點並點選「User Settings」，指定 Maven 設定檔位置「D:\IDE\apache-maven-3.6.1\conf\settings.xml」：



▲ 圖 1-19 指定 Maven 設定檔

另外，執行 Maven 指令只有 JRE 不夠，必需要 JDK，因此要修改 Eclipse 設定使其使用 JDK 編譯 Java 原始碼。開啟「Preferences」視窗，展開「Java」節點並點選「Installed JREs」，須將路徑指定使用 JDK：



▲ 圖 1-20 指定使用 JDK 編譯 Java 原始碼

若非如此，後續使用 Maven 編譯 Java 時將出現以下失敗訊息：

結果

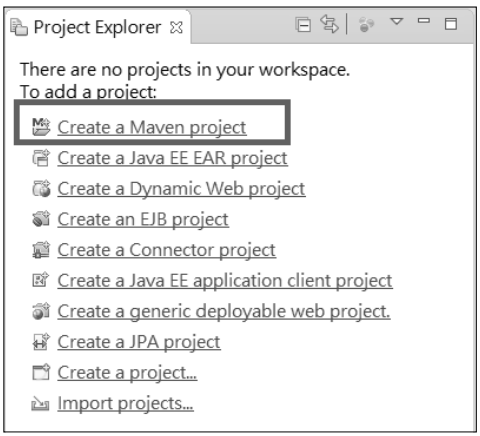
```
[ERROR] No compiler is provided in this environment. Perhaps you are running on a
JRE rather than a JDK?
```

1.3.2 使用 Eclipse 以 Maven 建立 Java SE 專案

建立 Eclipse 專案

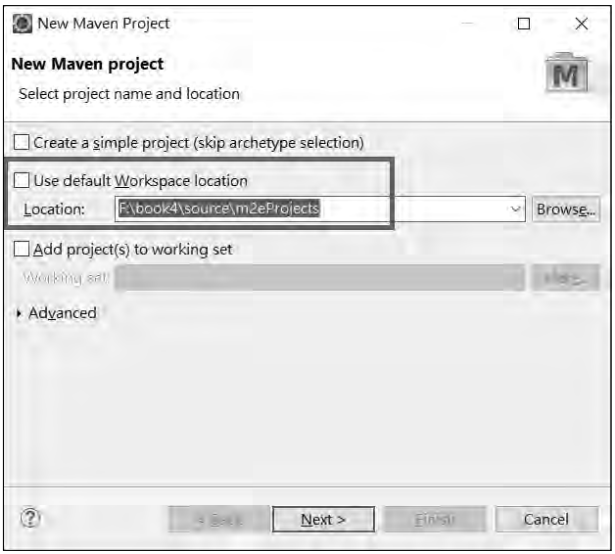
要以 Maven 管理 Java 專案除了使用指令外，也可以使用 Eclipse。步驟如下：

Step01 在 Project Explorer 的頁籤下，點擊「Create a Maven project」：



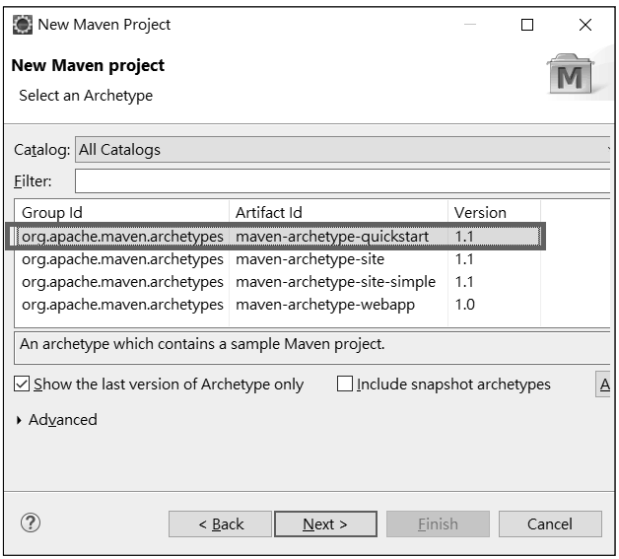
▲ 圖 1-21 Create a Maven project

Step02 指定新建的 Maven 專案位置，預設使用 Eclipse 的 workspace，或自己指定位置：



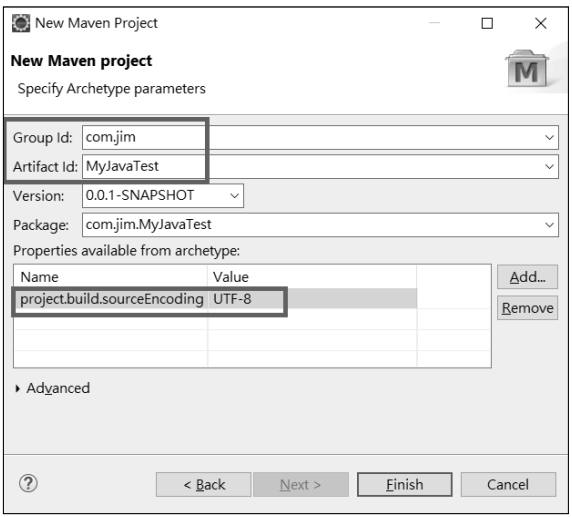
▲ 圖 1-22 指定 Maven 專案位置

Step03 選擇要使用的 Maven 模板。本例為建立 Java SE 專案故選擇「maven-archetype-quickstart」：



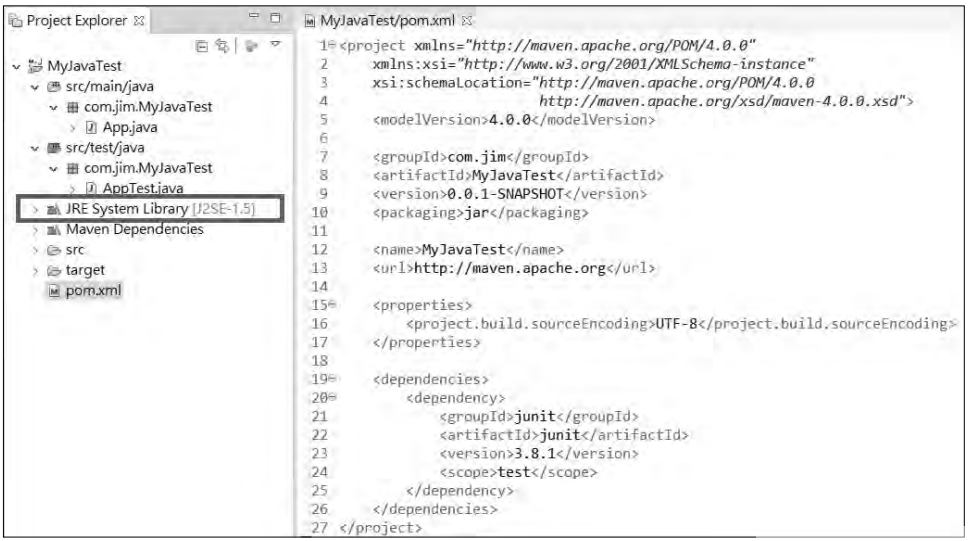
▲ 圖 1-23 指定 maven-archetype-quickstart

Step04 輸入建立 pom.xml 的必要資訊如「Group Id」和「Artifact Id」，再輸入其他屬性資訊如「project.build.sourceEncoding」並指定值「UTF-8」；或是產生 pom.xml 後再更改。完成後點擊按鍵 Finish：



▲ 圖 1-24 輸入建立 Maven 專案的資訊

Step05 使用 Maven 建立 Java SE 專案「MyJavaTest」完成後如下，蒐錄在本書範例專案資料夾 c01\m2eProjects\MyJavaTest：



▲ 圖 1-25 使用 Maven 完成 Java SE 專案