

安裝 Maven

- 首先安裝 Maven 本人。
 - windows 用戶請[參考此處](#)。
 - mac 用戶使用終端機介面，輸入指令 **brew install maven**，若還沒安裝 brew 請[參考此處](#)。
 - 可以的話換 mac 吧！
- 建議不要使用 IDE 的內建 Maven 工具，此手札範例全程皆以指令練習。
- 上緊發條，我們開始吧！

setting.xml 概念

- `${Maven Home}/conf` 的 `setting.xml` 為全域設置。
- `~/.m2/` 的 `setting.xml` 為該 User 設置。
- 若設置 User 範圍的 `setting.xml`，在升級 Maven 版本的時候就不需要更新 `setting.xml` 文件（建議的做法）。
- `setting.xml` 可設置元素請參考附圖。

元素名称	简介
<code>< settings ></code>	settings.xml 文档的根元素
<code>< localRepository ></code>	本地仓库
<code>< interactiveMode ></code>	Maven 是否与用户交互，默认值 true
<code>< offline ></code>	离线模式，默认值 false
<code>< pluginGroups ></code> <code>< pluginGroup ></code>	插件组
<code>< servers ></code> <code>< server ></code>	下载与部署仓库的认证信息
<code>< mirrors ></code> <code>< mirror ></code>	仓库镜像
<code>< proxies ></code> <code>< proxy ></code>	代理
<code>< profiles ></code> <code>< profile ></code>	Settings Profile
<code>< activeProfiles ></code> <code>< activeProfile ></code>	激活 Profile

pom.xml 基礎屬性

- POM 為 Project Object Model 的縮寫。
- `modelVersion`：POM 的版本，對 Maven 3 來說，只能是 4.0.0。
- `groupId`：com.googlecode.myapp（package name）。
- `artifactId`：可以稱為 project name（module name）。

- version：預設為 1.0-SNAPSHOT。其中 SNAPSHOT 為快照的意思，為不穩定的版本。
- name：非必要，但可以提供一個對開發者來說更友善的名稱辨識於主控台上。
- packaging：打包方式，預設為 jar。
- **Maven 座標 = groupId+artifactId+version**，對 Maven 來說這就是依賴的 ID。

Maven 的主要精神

慣例優先於配置，Maven 認為你不應花費時間在這些慣例的設定上，而要將精神放在更重要的工作上。於是有了以下約定...

1. src/main/java 主要程式的位置。
2. src/test/java 測試程式的位置。
3. 若沒有遵循 Maven 的慣例開發，在使用 plugin 時可能會有預期外的例外發生。
4. 若沒有遵循 Maven 的慣例開發，請問自己三次「我真的要這樣嗎？」。
5. 呈上述，最好不要。

Scope

- 針對 classpath，常用的三個設定如下
 - compile：預設值，對於編譯、測試以及執行都有效。
 - test：僅對測試有效。
 - provided：編譯以及測試有效。

依賴範圍 (Scope)	對於編譯 classpath 有效	對於測試 classpath 有效	對於運行時 classpath 有效	例 子
compile	Y	Y	Y	spring-core
test	—	Y	—	JUnit
provided	Y	Y	—	servlet-api
runtime	—	Y	Y	JDBC 驅動實現
system	Y	Y	—	本地的，Maven 倉庫之外的 的類庫文件

排除依賴

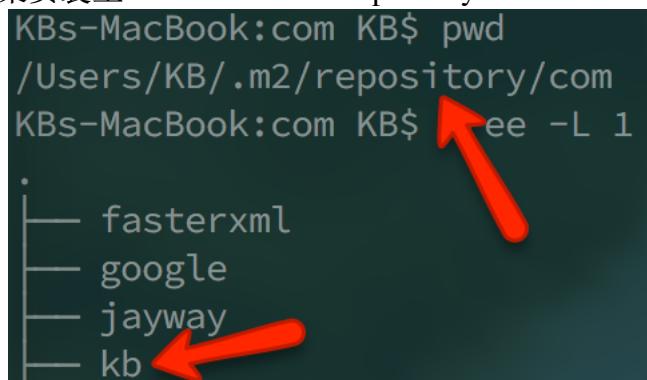
不用宣告 version，Maven 自動處理，在引入的依賴中排除指定的依賴。

```
<dependency>
  <groupId>com.juvenxu.mvnbook</groupId>
  <artifactId>project-b</artifactId>
  <version>1.0.0</version>
  <exclusions>
    <exclusion>
      <groupId>com.juvenxu.mvnbook</groupId>
      <artifactId>project-c</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Repository

- **Internal Repository**

- Maven 的檔案結構不會有 lib/ 存放依賴的檔案結構。
- 在預設情況下在本機端會有一個 .m2/repository 的依賴目錄。
- **mvn install** 會將該 Maven 專案安裝至 Internal Maven Repository。



A terminal window showing the command `pwd` and `tree -L 1` being executed in the directory `/Users/KB/.m2/repository/com`. The `tree` command lists the subdirectories: `fasterxml`, `google`, `jayway`, and `kb`. Two red arrows point to the `tree -L 1` command and the `kb` directory.

```
KBs-MacBook:com KB$ pwd
/Users/KB/.m2/repository/com
KBs-MacBook:com KB$ tree -L 1
.
├── fasterxml
├── google
├── jayway
└── kb
```

- 呈上述，install 結果如附圖。
- 呈上述，我們可以在程式打包後的輸出目錄 `/target/maven-archiver` 下找到 Maven 座標資訊（`groupId+artifactId+version`）即可使其它 Maven 專案參考。

- **Remote Repository**

- 對 Maven 專案來說，引入依賴的過程會先從 Internal 開始，如果沒有找到相關依賴，才會尋找 Remote（預設情況下會是中央庫）。
- 又可分為中央庫，或是私人庫 ex: 組織內部使用 Maven server。
- 私人用的 Maven server 會有以下優勢
 1. 節省對外網路頻寬（不用重複對中央庫擷取依賴並下載）
 2. 加速 Maven 建構速度（不需要一直重複檢查中央庫依賴版本）

3. 部署第三方依賴（如有版權問題沒有提供中央庫服務的狀況 like Oracle JDBC Driver）
 4. 提高穩定性（即使對外網路有問題，私人庫還是有既有的依賴可以使用）
 5. 降低中央庫的負載
- 有時候會有中央庫無法滿足的狀況，可能會尋求其他遠程庫，如 JBoss Maven 庫。
 - 呈上述，附圖為 JBoss 其中一個依賴庫參考示意圖。

```
<repositories>
  <repository>
    <id>jboss-enterprise-maven-repository</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

- releases：是否支援發布版本的依賴。
- snapshots：是否支援快照版本的依賴。
- 呈上述，該示意圖僅會下載 JBoss 發布版本的依賴，快照版本的不會。[參考此處](#)
- releases / snapshots 的共用屬性
 - updatePolicy：更新的策略，預設是每天 [never ,always ,interval:mins ,daily]。
 - checksumPolicy：Maven 檢核文件的策略。[ignore, fail ,warn]
- 在有必要的時後，我們可以替 Repository 添加認證屬性（id/username/password in setting.xml & POM）for 下載遠程庫依賴或是部署依賴至遠程庫，[參考此處](#)。
- 部署至遠程庫[參考此處](#)，通常會配合上述的認證設定。配置完成後使用 **mvn clean deploy** 啟動部署程序至指定的遠程庫。

SNAPSHOT

若為 SNAPSHOT 版本，在每次部署至 Repository 的時候，Maven 會偷偷加上時間戳記。

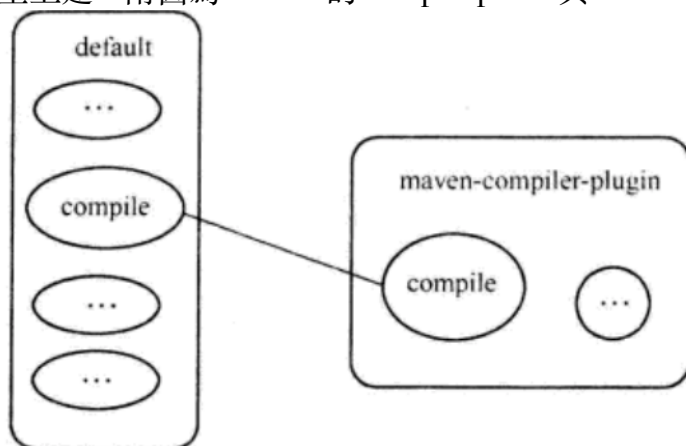
也就是說，只要不穩定的版本持續更新，在版本號沒有置換的情況下可以一直取得最新的不穩定版本。這種模式通常為組織內部開發使用。

Maven 簡易生命週期介紹

- 一個專案就是重複的執行「編譯、測試、打包、部署」，所以需要 Maven 幫我們簡化這些過程。
- Maven 擁有三套獨立的生命週期。[參考此處](#)。
- 透過多個 phase 組合成 lifecycle。
- phase 有順序性，如果調用某個 phase 則該 phase 以前的所有 phase 會一併被調用。
- **mvn** 指令的使用主要是調用 lifecycle 中的某個 phase。
 - ex **mvn clean** 調用 clean lifecycle 中的 clean phase。
 - ex **mvn test** 調用 default lifecycle 中的 test phase。

Plugin

- 使用時的概念為 **plugin:goal**，像是 compiler:compile 或是 dependency:tree。
- 在執行 **mvn plugin:goal** 的時候，如果 plugin 支援參數設置可透過 -D[property]=value 的方式帶入參數，運作原理為設置一個 java 的系統屬性並由插件讀取後執行指定的 goal。
- Maven 的 lifecycle 會與 plugin 綁定，用以完成實際的建構任務。
- 呈上述，具體而言為 Maven lifecycle phase <---> plugin goal。
- 呈上述，附圖為 default 的 compile phase 與 maven-compiler-plugin 綁定的示意圖。



- Maven 有很多預設的綁定 plugin，當 Maven 的 lifecycle 中某個 phase 被調用，此時綁定該 phase 的 plugin 便會去執行綁定的 goal。[參考此處](#)

- 附圖為 default lifecycle 各 phase 綁定的 default maven plugin 示意圖。

生命周期阶段	插件目标	执行任务
process-resources	maven-resources-plugin;resources	复制主资源文件至主输出目录
compile	maven-compiler-plugin;compile	编译主代码至主输出目录
process-test-resources	maven-resources-plugin;testResources	复制测试资源文件至测试输出目录
test-compile	maven-compiler-plugin;testCompile	编译测试代码至测试输出目录
test	maven-surefire-plugin;test	执行测试用例
package	maven-jar-plugin;jar	创建项目 jar 包
install	maven-install-plugin;install	将项目输出构件安装到本地仓库
deploy	maven-deploy-plugin;deploy	将项目输出构件部署到远程仓库

- 附圖為綁定 plugin 至 default lifecycle and phase=process-test-resources 的示意圖。

```
<plugin>
  <groupId>com.mycompany.example</groupId>
  <artifactId>display-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <id>display-time</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>time</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- id：將顯示於主控台辨識任務用。
- phase：將 plugin 綁定至某個指定的 phase。
- goal：plugin 將執行的 goal。
- 當多個 plugin 綁定到相同的 phase，則定義 plugin 的順序將決定執行的順序。
- 完整的 Plugins 列表請[參考此處](#)。

Plugin-surefire

- **mvn test** 需要注意的是，這個指令預設針對在 `src/test/java` 下的如附圖命名規範有

```
<includes>
  <include>**/Test*.java</include>
  <include>**/*Test.java</include>
  <include>**/*Tests.java</include>
  <include>**/*TestCase.java</include>
</includes>
```

效。

- 如果想跳過測試（再問自己三遍），可以執行以下指令 **mvn package -DskipTests**。
- `cd` 至指定的 Maven 專案目錄下後，可使用 **mvn test** 指令指定特定檔案測試。請參考如下範本
 - **mvn test -Dtest=className**
 - **mvn test -Dtest=Some*Test**
 - **mvn test -Dtest=class1,class2**
 - **mvn test -Dtest=class,Some*Test**
- 透過 plugin 設定的方式，指定或是排除特定命名測試，請[參考此處](#)。
- 測試報告可以在打包完之後的 `target/surefire-reports` 找到，分為 `.txt` 與 `.xml` 兩種格式。
- 在打包 Maven 專案的時候，若有需要一併打包測試代碼，請[參考此處](#)。

Aggregation (Multi-Module)

- 可將多個 Maven 專案一起執行 **mvn** 指令 [參考此處](#)。
- 慣例上多個同組織的專案 `groupId`、`version` 需要為一致的，在性質相同的專案 `artifactId` 前綴名字最好也能夠一致，如果有 `name` 屬性在主控台上將能更清晰的辨識專案。
- 呈上述，聚合用的 Maven 專案也需要遵循上述的慣例。
- 聚合用 Maven 專案的 `packaging` 必為 **pom**。
- 結構不一定是父子關係，可以為平行的目錄參考如附圖。

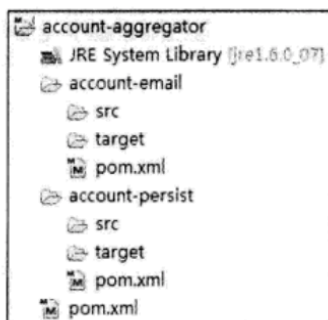


图 8-1 聚合模块的父子目录结构

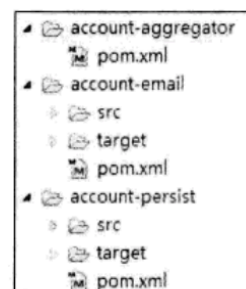


图 8-2 聚合模块的平行目录结构

- 呈上述，差別為在 modules 下的 module 的參考目錄需要做調整 (../)。
- 附圖為基本的聚合用平行 Maven 專案參

```
<groupId>com.kb.maven.demo</groupId>
<artifactId>ctbcAggregator</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
<name>CTBC Aggregator</name>

<modules>
  <module>../ctbcParent</module>
  <module>../ctbcCore</module>
  <module>../ctbcRestful</module>
</modules>
```

考。

- 呈上述，mvn clean package 執行結果局部如附圖。

```
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] CTBC Parent ..... SUCCESS [ 0.160 s]
[INFO] CTBC Core ..... SUCCESS [ 1.914 s]
[INFO] CTBC Restful ..... SUCCESS [ 0.457 s]
[INFO] CTBC Aggregator ..... SUCCESS [ 0.003 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

- 若有特殊建構需求，如跳過某個指定的模組加速建構（ex: 建構非常耗時的模組），請[參考此處](#)。

Parent

- 類似 OO 的繼承概念，解決重複性代碼（groupId、version、dependency or plugin ...）。
- 同 aggregation，packaging 必為 pom。
- 與子模組一致的 groupId、version，設定請參考附圖。

```
<groupId>com.kb.maven.demo</groupId>
<artifactId>ctbcParent</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
<name>CTBC Parent</name>
```


- 子模組繼承的方式請參考附圖。

```
<parent>
  <groupId>com.kb.maven.demo</groupId>
  <artifactId>ctbcParent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <relativePath>../ctbcParent/pom.xml</relativePath>
</parent>
```

- 呈上述，除了 parent pom 基本座標以外，必須透過 relativePath 屬性明確定義 parent pom 的位置，因為 Maven 預設會去查找 **./pom.xml**（父子結構），非平行結構。
- 子模組繼承了 parent pom 以後，無需再定義 groupId & version，這些屬性已經被繼承了，若有必要可再明確的宣告。可被繼承的屬性請[參考此處](#)。

- 使用 **dependencyManagement**

- 集中管理 dependency。
- 子模組引入依賴僅需要 groupId、artifactId，version 由 parent pom 集中管理。
- 降低各依賴版本衝突的可能性。
- 子模組如果沒有宣告任何依賴，也不會引入在 parent pom dependencyManagement 定義的任何依賴。
- 也可以宣告依賴的 scope，子模組也可以繼承，請參考附圖。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
      <version>${junit.version}</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- 如果想要在 A 模組引入與 B 模組一模一樣的 dependencyManagement 配置，也可以使用如附圖的方式。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-javaee7</artifactId>
      <version>${jboss.eap.bom.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- 呈上述，scope 使用的是 import，此特性僅在 dependencyManagement 下有效果，必須設定 **type=pom**。

- 使用 **pluginManagement**

- 集中管理 plugin。
- 若子模組需要的 plugin 與 parent pom 的 pluginManagement 設定的一致，則無須再宣告任何 plugin。
- 呈上述，這是與 dependencyManagement 比較不一樣的地方，dependencyManagement 如果要引用依賴必須明確的宣告 groupId、artifactId。
- 如果子模組不需要 parent pom 設定的 plugin，可以忽略它們。
- 如果子模組需要不同的 plugin 組態，可以自行配置覆蓋過 parent pom 的設定。
- 需要注意的是若子模組需要不同的組態，也應使用 parent pom 所設定的 version，統一 plugin 的 version 降低潛在的錯誤的發生機率。
- 設定方式請參考附圖。

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <encoding>${ctbc.encoding}</encoding>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
          <encoding>${ctbc.encoding}</encoding>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

Aggregation vs Parent

- 結構上無太大差別，都不需要任何的實作。但是用途不一樣。
- 實際項目中，aggregation pom & parent pom 會為了方便而併在一起使用。



- 關係請參考附圖。

- aggregation 主要用來集中管理或是執行 Maven 任務，這些 Maven 專案並不知道自己正在被管理。
- parent 主要用來消除重複的程式碼，繼承的 Maven 專案必須知道 parent pom 在哪裡。

Maven Web

- Java Web 的應用標準打包檔副檔名為 .war。
- 在 war 的結構底下，應會有 WEB-INF 的資料夾，裡面有 classes & lib。
- Maven Web 必須指定 packaging 為 **war**。
- Maven Web 除了預設的約定資料夾，會多一個 webapp 的約定資料夾，路徑 **src/main/webapp**。
- 在 Servlet 3.0 以前 WEB-INF 裡必須有一個需要有一個 web.xml 的配置。
- Maven Web 配置請參考附圖。

圖一：

```
<artifactId>ctbcRestful</artifactId>
<packaging>war</packaging>
<name>CTBC Restful</name>

<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>ctbcCore</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
      <exclusion>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>

<build>
  <finalName>ctbc</finalName>
</build>
```

圖二：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.0.0</version>
  <configuration>
    <failOnMissingWebXml>${ctbc.enableWebXml}</failOnMissingWebXml>
  </configuration>
</plugin>
```

- 圖一宣告 packaging = war。
- 圖一的 dependency 的 ctbcCore 的座標資訊參考到 project properties，因為他們繼承了同樣的 parent pom（不希望篇幅太多沒有貼出全部原始碼請見諒）。
- 圖一宣告 build > finalName = ctbc，所以打包出來的 war 檔為 ctbc.war。
- 圖二的 maven-war-plugin 插件提供了一個是否在「沒有 web.xml 的時候錯誤」的屬性，為 Servlet 3.0 以上版本帶來彈性。
- 使用 Cargo plugin 自動化部署
 - 支援多個 Web Container 部署。請[參考此處](#)。
 - 範例為針對 Tomcat 8 的部署，情境為有一個 Tomcat 8 正在 localhost:8080 作用中。
 - 請參考如附圖設定。

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.6.3</version>
  <configuration>
    <container>
      <containerId>tomcat8x</containerId>
      <type>remote</type>
    </container>
    <configuration>
      <type>runtime</type>
      <properties>
        <cargo.remote.username>${tomcat.user}</cargo.remote.username>
        <cargo.remote.password>${tomcat.password}</cargo.remote.password>
        <cargo.tomcat.manager.url>${tomcat.manager.url}</cargo.tomcat.manager.url>
      </properties>
    </configuration>
  </configuration>
</plugin>
```

- 作用中的 Web Container 在 container > type 必需設定為 remote。
- 作用中的 Web Container 在 configuration > type 必需設定為 runtime。
- 在 configuration > properties 必須設定 Tomcat 部署需要用到的資訊（使用者帳號、密碼以及 Manager Url）。
- 需要注意的是該使用者必須擁有 manager-script 的使用者權限（/tomcat/conf/tomcat-users.xml）。
- 權限設定部分請參考附圖。

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin" roles="manager-gui,manager-script"/>
```

- cargo tomcat8x 相關細部設定請[參考此處](#)。

- 在 .m2 設定 setting.xml 請參考附圖。

```
KBs-MacBook:~ KB$ pwd
/Users/KB/.m2
KBs-MacBook:~ KB$ cat setting.xml
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <pluginGroups>
    <pluginGroup>org.codehaus.cargo</pluginGroup>
  </pluginGroups>
</settings>
```

- 如果忘記 setting.xml 是什麼請到文章頂複習一下並敲自己頭三下。
- 呈上述，設定的理由為「只有 **org.apache.maven.plugins** 、**org.codehaus.mojo** 這兩個 groupId 下的 plugins 才能簡化 command line」，設定完之後才可以使用 **mvn cargo:goal**。
- 執行 **mvn clean package cargo:redploy**。
- 呈上述，**cargo:redploy** 的作用為部署程式至 server 上，若 server 上有相同應用則將其卸載後重新部署。

Maven 版本約定

- 請參考附圖說明。

<主版本> . <次版本> . <增量版本> - <里程碑版本>

主版本和次版本之间，以及次版本和增量版本之间用点号分隔，里程碑版本之前用连字号分隔。下面解释其中每一个部分的意义：

- ❑ **主版本**：表示了项目的重大架构变更。例如，Maven 2 和 Maven 1 相去甚远；Struts 1 和 Struts 2 采用了不同的架构；JUnit 4 较 JUnit 3 增加了标注支持。
- ❑ **次版本**：表示较大范围的功能增加和变化，及 Bug 修复。例如 Nexus 1.5 较 1.4 添加了 LDAP 的支持，并修复了很多 Bug，但从总体架构来说，没有什么变化。
- ❑ **增量版本**：一般表示重大 Bug 的修复，例如项目发布了 1.4.0 版本之后，发现了一个影响功能的重大 Bug，则应该快速发布一个修复了 Bug 的 1.4.1 版本。
- ❑ **里程碑版本**：顾名思义，这往往指某一个版本的里程碑。例如，Maven 3 已经发布了很多里程碑版本，如 3.0-alpha-1、3.0-alpha-2、3.0-beta-1 等。这样的版本与正式的 3.0 相比，往往表示不是非常稳定，还需要很多测试。

需要注意的是，不是每个版本号都必须拥有这四个部分。一般来说，主版本和次版本都会声明，但增量版本和里程碑就不一定了。例如，像 3.8 这样的版本没有增量和里程碑，2.0-beta-1 没有增量。

Dynamic Boot

- Maven 有許多預設的變數可幫助我們建構專案，請[參考此處](#)。
- 使用 **mvn help:system** 查看可以使用的變數，並在 pom 檔內透過 `${ var }` 的方式控制它們。請[參考此處](#)。
- 善用變數可以降低在版本升級或是環境遷移時需要更改參數的數量。
- 一些 plugin 預設也會使用變數當作參數決定行為，請[參考此處](#)。
- **Profile**
 - 針對不同環境的配置，可能某些環境需要特定的依賴、屬性或是特殊的插件。
 - 可以透過 **mvn clean package -P[profileId]** 啟動指定的 profile，範例如下
 - **mvn clean package -Psit**
 - **mvn clean package -Psit-x,sit-y**（多個 profile 以逗號分隔）
 - **mvn help:active-profiles** 列出正在作用中的 profiles
 - **mvn help:all-profiles** 列出當前所有的 profiles
 - 可配置的屬性以及詳細說明請[參考此處](#)。
- **POM extends Spring Boot Parent & Profile** 過濾設定檔範例請參考附圖配置

圖一：

```
<profiles>
  <profile>
    <id>local</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <env.current>local</env.current>
      <env.jdbc.driverClassName>com.mysql.jdbc.Driver</env.jdbc.driverClassName>
      <env.jdbc.userName>root</env.jdbc.userName>
      <env.jdbc.password>root123</env.jdbc.password>
    </properties>
  </profile>
  <profile>
    <id>sit</id>
    <properties>
      <env.current>sit</env.current>
      <env.jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</env.jdbc.driverClassName>
      <env.jdbc.userName>admin</env.jdbc.userName>
      <env.jdbc.password>admin123</env.jdbc.password>
    </properties>
  </profile>
</profiles>
```

```
<resources>
  <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
  </resource>
</resources>
```

圖二：


```
env.current=@env.current@

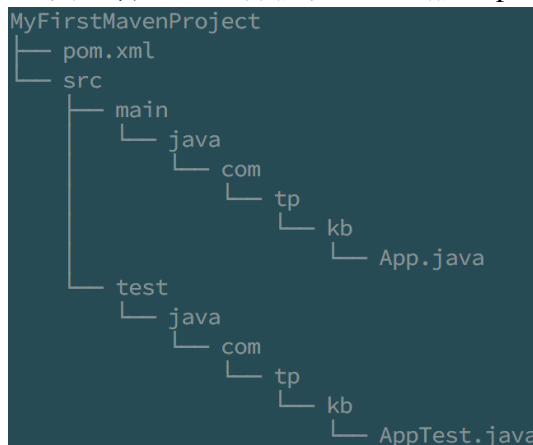
#connection
env.jdbc.driverClassName=@env.jdbc.driverClassName@
env.jdbc.userName=@env.jdbc.userName@
env.jdbc.password=@env.jdbc.password@
```

圖三：

- 圖一宣告了兩個 profile，分別為 local、sit，屬性設定也不一樣。
- 圖一的 local profile 透過 activation > activeByDefault = true 宣告為預設的 profile。
- 圖二開啟了資源過濾，會將 pom 設定的屬性置換到指定的目錄底下的檔案，此範例置換符為 @ * @。
- 圖三為 jdbc.properties 連線資訊設定檔，打包後屬性值的部分將會被置換。
- 分別執行 **mvn clean package**、**mvn clean package -Psit** 觀察打包後的檔案。
- 注意：預設置換符應為 \${ * }，而不是 @ * @，因為 Spring Boot Parent 更改了置換符。請[參考此處](#)，以及參閱本手札隨筆章節的附圖。
- Maven Web 資源的過濾請[參考此處](#)。

常用指令集清單

- **mvn -v**：取得機器目前的 Maven 版本以及相關資訊。
- **mvn help:system**：印出所有 java 系統屬性以及環境變數。
- **mvn help:describe**：印出插件相關訊息 [參考此處](#)。
- **mvn help:active-profiles**：列出正在作用中的 profiles。
- **mvn help:all-profiles**：列出當前所有的 profiles。
- **mvn dependency:list**：印出依賴清單。
- **mvn dependency:tree**：印出依賴樹。
- **mvn dependency:analyze**：分析依賴狀態，主要分析主程式以及測試程式用到的依賴。
 - Used undeclared dependencies：使用未明確定義的依賴。
 - Unused declared dependencies：沒有使用但明確定義的依賴。
- **mvn archetype:generate**：自動生成 Maven 檔案結構。
 - 附圖包含一支執行檔以及測試檔，pom.xml 會引入一個 junit 的依賴。



- **mvn clean**：清除預設輸出目錄 target/ (clean:clean)
- **mvn compile**：編譯主要目錄程式 src/main/java 輸出至 target/classes (resources:resources & compiler:compile)
- **mvn test**：執行測試代碼 src/test/java with junit (resources:resources & compiler:compile & resources:testResources & compiler:TestCompile & surefire:test)
- **mvn package**：打包程式碼，預設輸出為 .jar 檔，檔名為 artifactId-version.jar (resources:resources & compiler:compile & resources:testResources & compiler:TestCompile & surefire:test & jar:jar)
- **mvn install**：安裝至 Internal Repository，使其他的 Maven 項目可以參考 (resources:resources & compiler:compile & resources:testResources & compiler:TestCompile & m & jar:jar & install:install)

隨筆：

在執行 **mvn clean test** 的時候，因為 Maven 的 compiler 插件默認支援 Java1.3，當使用高於 java1.3 的特性的時候就會報錯，所以通常我們需要加入以下 plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- 若是繼承 Spring Boot Parent，請[參考此處](#)的解法（僅需設定 java.version 的屬性即可）。

- 呈上述，maven-resources-plugin 除非有特殊需求，不然也會使用 Spring Boot Parent 預設的 UTF-8。

請參考附圖屬性設定（參考至 org.springframework.boot:spring-boot-starter-parent:1.5.4.RELEASE）。

```
<properties>
  <java.version>1.6</java.version>
  <resource.delimiter>@</resource.delimiter> <!-- delimiter that doesn't clash with Spring ${} placeholders -->
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <maven.compiler.target>${java.version}</maven.compiler.target>
</properties>
```

建立時間：2017/06/16

完成時間：2017/06/22

來源參考：網路資源、[Maven 實戰](#)

作者：KB.Liao

信箱：kb19900709@gmail.com