Java SE 7 新功能索引

chapter

1 Java 平台概論

學習目標

- 簡介 Java 版本遷移
- 認識 Java SE、Java EE、Java ME
- 瞭解 JVM、JRE 與 JDK
- 下載、安裝 JDK

1.1 Java 不只是語言

從 1995 年至今，Java 已經超過 15 個年頭，經過這些年來的演進，正如本節標題所示，Java 已不僅是個程式語言，也代表瞭解決問題的平台（Platform），更代表了原廠、各個廠商、社群、開發者與使用者溝通的成果。若僅以程式語言的角度來看待 Java，正如冰山一角，你僅看到 Java 身為程式語言的一部份，而沒看到 Java 身為程式語言之外，更可貴也更為龐大的資源。

1.1.1 前世今生

一個語言的誕生有其目的，因為這個目的而成就了語言的主要特性，探索 Java 的歷史演進，對於掌握 Java 特性與各式可用資源，著實有其幫助。

Java 誕生

Java 最早是 Sun 公司「綠色專案」（Green Project）中撰寫 Star7 應用程式的程式語言，當時名稱不是 Java，而是取名為 Oak。

綠色專案始於 1990 年 12 月，由Patrick Naughton、Mike Sheridan與James Gosling[1] 主持，目的是希望構築出下一波電腦應用趨勢並加以掌握，他們認為下一波電腦應用趨勢會集中在消費性數位產品（像是今日的PDA、手機等消費性電子商品）的使用上，在 1992 年 9 月 3 日Green Team專案小組展示了Star7 手持設備，這個設備具備無線網路連接、5 吋LCD彩色螢幕、PCMCIA介面等功能，而Oak在綠色專案中的目的，是用來撰寫Star7 上應用程式的程式語言。

Oak 名稱的由來，是因為 James Gosling 的辦公室窗外有一顆橡樹（Oak），就順手取了這個名稱。但後來發現 Oak 名稱已經被註冊了，工程師們邊喝咖啡邊討論著新名稱，最後靈機一動而改名為 Java。

[1] James Gosling 被尊稱為 Java 之父。

Java 本身會見到許多為了節省資源而作的設計，像是動態載入類別檔案、字串池（String pool）等特性，這是因為 Java 一開始就是為了消費性數位產品而設計，而這類小型裝置通常有著有限記憶體與運算資源。

全球資訊網（World Wide Web）興起，Java Applet 成為網頁互動技術代表。

1993 年第一個全球資訊網瀏覽器 Mosaic 誕生，James Gosling 認為網際網路與 Java 的一些特性不謀而合，利用 Java Applet 在瀏覽器上展現互動性媒體，在當時而言，對視覺感官是一種革命性的顛覆，Green Team 仿照 Mosaic 開發出以 Java 技術為基礎的瀏覽器 WebRunner（原命名為 BladeRunner），後來改名為 HotJava，雖然 HotJava 只是一個展示性產品，但它使用 Java Applet 展現的多媒體效果馬上吸引許多人的注意。

1995 年 5 月 23 日 [2]，正式將Oak改名為Java，Java Development Kits（當時JDK全名）1.0a2 版本正式對外發表，而在 1996 年Netscape Navigator 2.0 也正式支援Java，Microsoft Explorer亦開始支援Java，從此Java在網際網路的世界中逐漸風行起來，雖然Star7 產品並不被當時消費性市場接受，綠色專案面臨被裁撤的命運，然而全球資訊網（World Wide Web）的興起卻給了Java新的生命與舞台。

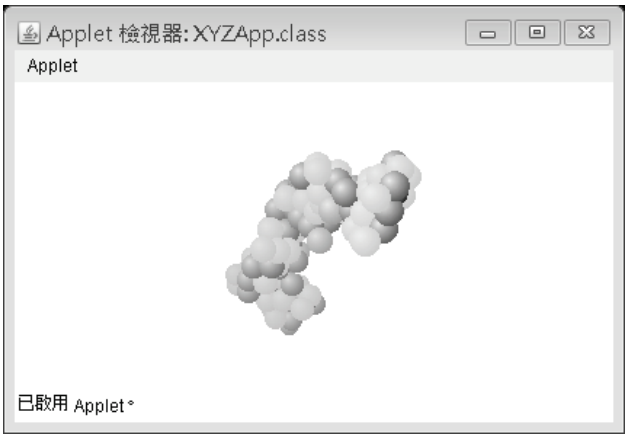

圖 1.1. JDK 所附的 Java Applet 範例
（JDK 資料夾\demo\applets\MoleculeViewer\ example1.html）

[2] 這一天公認為 Java 的誕生日。

版本演進

隨著 Java 越來越受到矚目，Sun 在 1998/12/4 年發佈 **Java 2 Platform**，簡稱 **J2SE 1.2**，Java 開發者版本一開始是以 **Java Development Kit** 名稱發表，簡稱 **JDK**，而 J2SE 則是平台名稱，包含了 JDK 與 Java 程式語言。

Java 平台標準版約每兩年為週期推出重大版本更新，1998/12/4 發表 J2SE 1.2，2000/5/8 發表 J2SE 1.3，2002/2/13 發表 J2SE 1.4，「Java 2」這個名稱也從 J2SE 1.2 一直延用至之後各個版本。

2004/9/29 發表的 Java 平台標準版的版號不是 1.5，而直接跳到 5.0，稱為 J2SE 5.0，這是為了彰顯這個版本與之前版本有極大不同，像是語法上的簡化、增加泛型（Generics）、列舉（Enum）、標註（Annotation）等重大功能。

2006/12/11 發表的 Java 平台標準版，除了版號之外，名稱也有了變化，稱為 **Java Platform, Standard Edition 6**，簡稱 **Java SE 6**，JDK6 全名則稱為 **Java SE Development Kit 6**，也就是不再像以前 Java 2 帶有"2"這個號碼，版本號 6 或 1.6.0 都使用，6 是產品版本（Product version），而 1.6.0 是開發者版本（Developer version）。

大部份的 Java 標準版平台都會取個代碼名稱（Code name），例如 J2SE 5.0 的代碼名稱為 Tiger（老虎），為了引人注目，在發表會上還真的抱了一隻小白老虎出來作為噱頭，而許多書的封面也應景地放上老虎的圖片。有關 JDK 代碼名稱與釋出日期，可以參考下表：

表 1.1 Java 版本、代碼名稱與釋出日期

版本	代碼名稱	釋出日期
JDK 1.1.4	Sparkler（煙火）	1997/9/12
JDK 1.1.5	Pumpkin（南瓜）	1997/12/3
JDK 1.1.6	Abigail（聖經故事人物名稱）	1998/4/24
JDK 1.1.7	Brutus（羅馬政治家名稱）	1998/9/28
JDK 1.1.8	Chelsea（足球俱樂部名稱）	1999/4/8
J2SE 1.2	**Playground（遊樂場）**	**1998/12/4**
J2SE 1.2.1	無	1999/3/30
J2SE 1.2.2	Cricket（蟋蟀）	1999/7/8
J2SE 1.3	**Kestrel（紅隼）**	2000/5/8

版本	代碼名稱	釋出日期
J2SE 1.3.1	Ladybird（瓢蟲）	2001/5/17
J2SE 1.4.0	**Merlin（魔法師名稱）**	**2002/2/13**
J2SE 1.4.1	Hopper（蚱蜢）	2002/9/16
J2SE 1.4.2	Mantis（螳螂）	2003/6/26
J2SE 5.0	**Tiger（老虎）**	**2004/9/29**
Java SE 6	**Mustang（野馬）**	**2006/12/11**
Java SE 7	**Dolphin（海豚）**	**2011/7/28**

提示 撰寫本書時，表 1.1 參考的資料來源為：
http://java.sun.com/j2se/codenames.html

江山易主

之前談過，Java 約以兩年為週期推出重大版本更新，正如表 1.1 所示，J2SE 1.2、J2SE 1.3、J2SE 1.4.0、J2SE 5.0、Java SE 6 推出的時間，差不多都是兩年，然後從 Java SE 6 之後，Java 開發人員足足等了四年多，才等到新版本的推出，不禁讓人想問：Java 怎麼了？

原因有許多，Java SE 7 對新版本的規劃搖擺不定，涵蓋許多不易實現的新特性，加上 Sun 一直苦於營收低迷不振，影響了新版本的推動，新版本推出日期承諾不斷跳票，從 2009 年推遲至 2010 年初，又突然宣佈將加入原本不願劃入 Java SE 7 的 Closure 語法，並將推出日期推遲至 2010 年底，然後 2010 年中傳出 IBM 與 Sun 密談併購失敗，沒隔幾日，即爆出 **Oracle** 宣佈併購 Sun，**Java 也正式成為 Oracle 所屬**。

併購就會帶來一連串的組織重整，導致 Java SE 7 推出日期再度跳票，為了對停滯不前的 Java 注入活水，決定先將現有已實現或較易實現的特性放入 Java SE 7 中，將未定案或較難實現的特性放入 Java SE 8 中（像是 Jigsaw、Closure），2010 年底 JCP（Java Community Process，稍後即會說明這個組織為何）終於通過了 Java SE 7 與 Java SE 8 的規劃地圖（Roadmap），並預計於 2011/7 左右推出 Java SE 7，這次總算沒有跳票，Java SE 7 正式於 2011/7/28 釋出。

提示 正因為 Java SE 7 推出過程如此曲折，網路上很容易找到過時的 Java SE 7 資料。如果你想快速瞭解 Java SE 7 有哪些特性，本書有標示出哪些特性是 Java SE 7 所有，亦有個快速查詢索引可作參考。

1.1.2 三大平台

在 Java 發展的過程中，由於 Java 的應用領域越來越廣，並逐漸擴及至各級應用軟體的開發，Sun 公司在 1999 年 6 月美國舊金山的 Java One 大會上，公佈了新的 Java 體系架構，該架構根據不同級別的應用開發區分了不同的應用版本：J2SE（Java 2 Platform, Standard Edition）、J2EE（Java 2 Platform, Enterprise Edition）與 J2ME（Java 2 Platform, Micro Edition）。

J2SE、J2EE 與 J2ME 是當時的名稱，由於 Java SE 6 後 Java 不再帶有"2"這個號碼，J2SE、J2EE 與 J2ME 分別被正名為 **Java SE**、**Java EE** 與 **Java ME**。

提示 儘管 Sun 從 2006 年底，就將三大平台正名為 Java SE、Java ME 與 Java EE，但時至今日，許多人的習慣顯然還是沒有改過來，J2SE、J2ME 與 J2EE 這個名詞還是很多人用。

Java SE（Java Platform, Standard Edition）

Java 各應用平台的基礎，想要學習其它的平台應用，必先瞭解 Java SE 以奠定基礎，Java SE 也正是本書主要的介紹對象。

下圖是整個 Java SE 的組成概念圖：

圖 1.2. Java SE 的組成概念圖

Java SE 可以分作四個主要的部份：**JVM**、**JRE**、**JDK** 與 **Java** 語言。

為了要能運行 Java 撰寫好的程式，必須有 **Java 虛擬機器（Java Virtual Machine, JVM）**。JVM 包括在 **Java 執行環境（Java SE Runtime Environment, JRE）**中，所以為了要運行 Java 程式，必須安裝 JRE。如果要開發 Java 程式，必須取得 **JDK（Java SE Development Kits）**，JDK 包括 JRE 以及開發過程中需要的一些工具程式，像是 javac、java、appletviewer 等工具程式（關於 JRE 及 JDK 的安裝與使用介紹，會在第 2 章說明）。

Java 語言只是 Java SE 的一部份，除了語言之外，Java 最重要的就是提供龐大且強大的標準 API，提供字串處理、資料輸入輸出、網路套件、使用者視窗介面等功能，你可以使用這些 API 作為基礎來進行程式開發，無須重複開發功能相同的元件，事實上，在熟悉 Java 語言之後，更多的時候，都是在學習如何使用 Java SE 提供的 API 來組成應用程式。

Java EE（Java Platform, Enterprise Edition）

Java EE 以 Java SE 為基礎，定義了一系列的服務、API、協定等，適用於開發分散式、多層式（Multi-tiered）、以元件為基礎、以 Web 為基礎的應用程式，整個 Java EE 的體系是相當龐大的，比較為人熟悉的技術像是 JSP、

Servlet、JavaMail、Enterprise JavaBeans（EJB）等，當中每個服務或技術都可以使用專書進行說明，所以並非本書說明的範圍，但可以肯定的是，必須在 Java SE 上奠定良好的基礎，再來學習 Java EE 的開發。

Java ME（Java Platform, Micro Edition）

Java ME 是 Java 平台版本中最小的一個，目的是作為小型數位設備上開發及部署應用程式的平台，像是消費性電子產品或嵌入式系統等，最為人熟悉的設備如手機、PDA、股票機等，你可以使用 Java ME 來開發出這些設備上的應用程式，如 Java 遊戲、股票相關程式、記事程式、月曆程式等。

1.1.3 JCP 與 JSR

Java 不僅是程式語言，而是標準規範！

先來看看沒有標準會有什麼問題？我們的身邊有些東西沒有標準，例如手機充電器，不同廠商的手機，充電器就不相同，家裡面一堆充電器互不相容，換個手機，充電器就不能用的情況，相信你我都有過！

有標準的好處是什麼？現在許多電腦週邊設備，都採用 USB 作為傳輸介面，這讓電腦中不用再接上一些轉接器，跟過去電腦主機後面一堆不同規格的傳輸介面相比，實在方便了不少（現在有些手機的充電器，也改採用 USB 介面了，這真是件好事）。

回頭來談談 Java 是標準規範這件事。你知道嗎？編譯/執行 Java 的 JDK/JRE，並不只有 Sun 才能實現，IBM 也可以撰寫自己的 JDK/JRE，其它廠商或組織也可以撰寫自己的 JDK/JRE，你寫的 Java 程式，可以執行在這些不同廠商或組織寫出來的 JRE 上。下一章將學到的第一個 Java 程式為例，其中會有這麼一段程式碼：

```
System.out.println("Hello World");
```

這行程式目的是：「請系統（System）的輸出裝置（out）顯示一行（println）"Hello World"」。誰決定使用 **System**、**out**、**println** 這些名稱的？為什麼不是 Platform、Output、ShowLine 這些名稱？如果 Sun 使用 System、out、println 這些名稱，而 IBM 使用了 Platform、Output、ShowLine 這些名稱，用 Sun 的

JDK 寫的程式，就不能執行在 IBM 的 JRE 上，那 Java 最基本的特性之一「跨平台」，就根本無法實現了！

Java 由 Sun 創造，為了讓對 Java 興趣的廠商、組織、開發者與使用者參與定義 Java 未來的功能與特性，Sun 公司於 1998 年組成了 **JCP（Java Community Process）**，這是一個開放性國際組織，目的是讓 Java 演進由 Sun 非正式地主導，成為全世界數以百計代表成員公開監督的過程。

任何想要提議加入 Java 的功能或特性，必須以 **JSR（Java Specification Requests）**正式文件的方式提交，JSR 必須經過 JCP 執行委員會（Executive Committee）投票通過，方可成為最終標準文件，有興趣的廠商或組織可以根據 JSR 實現產品。

若 JSR 成為最終文件後，必須根據 JSR 實作出免費且開發原始碼的參考實現，稱為 **RI（Reference Implementation）**，並提供 **TCK（Technology Compatibility Kit）**作為技術相容測試工具箱，方便其它想根據 JSR 實現產品的廠商或組織參考與測試相容性。

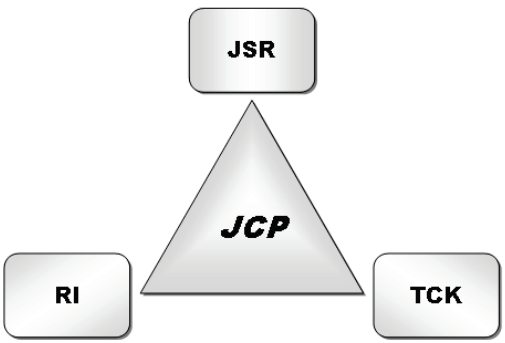

圖 1.3 JCP、JSR、RI **與** TCK

提示》》 JCP 官方網站為 http://jcp.org。

現在無論是 Java SE、Java EE 或 Java ME，都是業界共同訂製的標準，每個標準背後代表了業界所面臨的一些問題，他們期待使用 Java 來解決問題，認為應該有某些元件、特性、應用程式介面等，來解決這些問題，因而製訂 JSR 作為正式標準規範文件，不同的技術解決方案標準規範會給予一個編號。

在 JSR 規範的標準之下，各廠商可以各自實作成品，所以同一份 JSR，可以有不同廠商的實作產品，以 Java SE 為例，對於身為開發人員，或使用 Java 開發產品的公司而言，只要使用相容於標準的 JDK/JRE 開發產品，就可以執行於相容於標準的 JRE 上，而不用擔心跨平台的問題。

Java SE 7 的主要規範是在 **JSR 336** 文件之中，而 Java SE 平台中的特定技術，則再規範於特定的 JSR 文件之中，若對這些文件有興趣，可以參考以下網址：

- http://jcp.org/en/jsr/detail?id=336

提示 想要查詢 JSR 文件，只要在「http://jcp.org/en/jsr/detail?id=」加上文件編號就可以了，例如上面查詢 JSR 336 文件網址就是：

http://jcp.org/en/jsr/detail?id=336

JSR 對於 Java 初學者而言過於艱澀，但 JSR 文件規範了相關技術應用的功能，將來有能力時，可以試著自行閱讀 JSR，這有助於瞭解相關技術規範的更多細節。

1.1.4 建議的學習路徑

Java 不僅是程式語言，而是標準規範，每個標準代表著廠商面臨的問題，代表著解決問題的方案，也因此，學習 Java，就等於在面臨各式問題如何解決，然而，這麼多的問題，沿生出如此多的解決方案，也因此對於初學 Java 的人，如同面臨滿載產品的龐大貨輪，不知從何開始，也不知將來何去何從。

提示 如果程式語言被比喻為一艘船，會是如何呢？這邊有篇有趣的文章：

http://compsci.ca/blog/if-a-programming-language-was-a-boat/

在 Java 的官方網站提供有一份 Java 技術觀念地圖（Java Technology Concept Map）的文件，這是份 PDF，可以在以下的網址下載：

- http://java.sun.com/new2java/javamap/intro.html

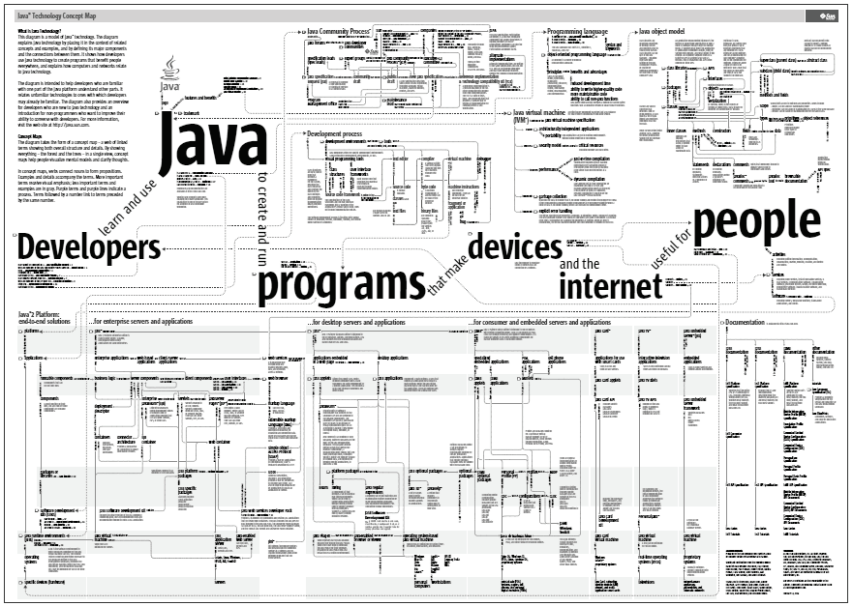

圖 1.4 Java 技術觀念地圖

在這份件中， 密密麻麻地列出了大部份 Java 相關地圖與簡要說明，也代表了 Java 技術範疇的廣泛，然而要從這麼龐大的地圖中，找出一條適合初學 Java 的路線圖絕非易事，以下是我基於經驗與教學建議的學習路徑：

深入瞭解 JVM/JRE/JDK

許多書籍對於 JVM/JRE/JDK 的說明，通常以極短的篇幅介紹，通常就是在短短幾頁中，請使用者依書中步驟安裝與設定 PATH、CLASSPATH 後，就開始介紹 Java 程式語言，而許多人到了業界後就開始使用 IDE（Integrated Development Environment）代勞所有 JDK 細節，這麼作的結果就是，在 IDE 中遇到與 JDK 相關的問題，就完全不知道如何解決。

JVM/JRE/JDK 並不是用短短幾頁就可以說明，若沒有「**JVM 是 Java 程式唯一認識的作業系統，其可執行檔為.class 檔案**」的重要觀念，就無法理解 PATH 與 CLASSPATH 並非同一層級的環境變數，JDK 中許多指令與選項，其實都可以對應至 IDE 中某個設定與操作，你對 JVM/JRE/JDK 有足夠的認識，對 IDE 中相關選項就不會有疑問，也不會換個 IDE 就不知所適，或沒有 IDE 就無法撰寫程式。

理解封裝、繼承、多型

對於 Java 程式語言，`if...else`、`for`、`while`、`switch` 等流程語法，早已是必須熟練的基礎，更重要的是，Java 支援物件導向（Object Oriented），你必須理解物件導向中最重要的**封裝（Encapsulation）**、**繼承（Inheritance）**、**多型（Polymorphism）**觀念，以及如何用 Java 相關語法來實現。

許多人撰寫 Java 程式，並沒有善用其支援物件導向的特性，問到何謂封裝而無法回答（甚至回答定義類別即為封裝），濫用 Java 繼承語法，不懂多型而不知如何運用 API 文件（更別說運用多型設計程式了），最後的結果就是淪於死背 API 文件、使用「複製、貼上」大法來撰寫程式，整個應用程式架構雜亂無章而難以維護。

掌握常用 Java SE API 架構

Java 並非只是程式語言，還帶有龐大的各式程式庫（Library），對初學者而言，首要是掌握常用的 Java SE API，像是例外（Exception）、群集（Collection）、輸入輸出串流（Stream）、執行緒（Thread）等，學習這些標準 API，絕不要淪於死背，應先**掌握 API 在設計時的封裝、繼承、多型架構**，以 `Collection` 為例，在學習時必須先理解，為何要設計為下圖的架構：

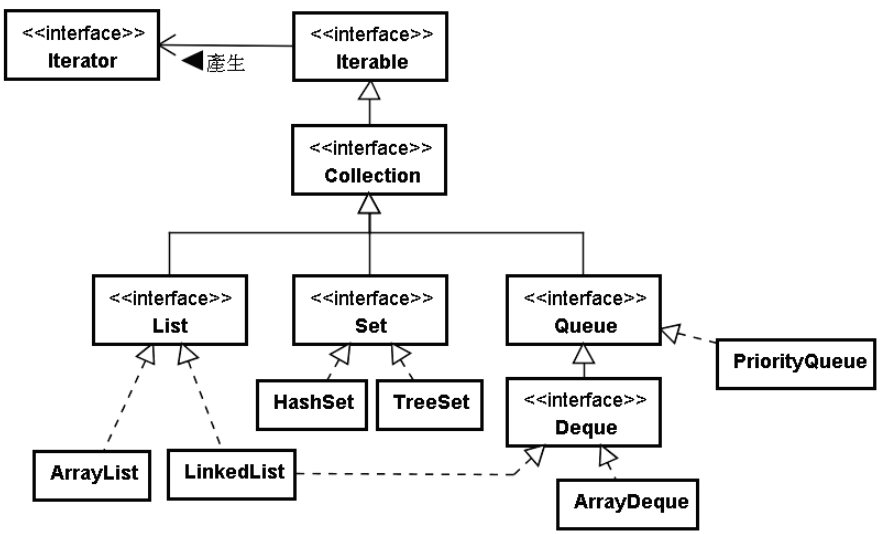

圖 1.5 Collection API 架構範例

學習相關程式庫或 API，先理解主要架構絕對是必要的，如此才不會淪於死背 API 或抄寫範例的窘境，更進一步地，還可以從 API 中學習到良好設計的觀念，有了這樣的好習慣，往後面對新的 API 或程式庫，就能更快掌握如何使用甚至改進。

提示 »» 「深入瞭解 JVM/JRE/JDK」、「理解封裝、繼承、多型」、「掌握常用 Java SE API 架構」都會是本書的說明重點。

學習容器觀念

在步入 Java EE 領域之後，經常接觸到**容器（Container）**的觀念，許多人完全以 API 層次來使用 Java EE 相關元件，這是不對的。容器就實作層面來說，就是執行於 JVM 上的 Java 應用程式，抽象層面來說，就是你的應用程式溝通、協調相關資源的系統。

初次接觸容器的開發人員會覺得容器很抽象，以實際的例子來說，通常初學者步入 Java EE，會從學習 Servlet/JSP 開始，而 Servlet/JSP 是執行於 Web 容器之中，這是學習容器觀念時不錯的開始，你必須知道**「Web 容器是 Servlet/JSP 唯一認得的 HTTP 伺服器，是使用 Java 撰寫的應用程式，運行於 JVM 之上」**，如果希望用 Servlet/JSP 撰寫的 Web 應用程式可以正常運作，就必須知道 Servlet/JSP 如何與 Web 容器作溝通，Web 容器如何管理 Servlet/JSP 的各種物件等問題。

提示 »» 關於 Servlet/JSP 的說明，可參考我撰寫的「Servlet/JSP 教學手冊第二版（碁峰出版社）」，或是 Servlet/JSP 線上文件：
http://caterpillar.onlyfun.net/Gossip/ServletJSP/

容器的觀念無所不在，Applet 會執行於 Applet 容器中，因此相關資源受到 Applet 容器的管理與限制，Servlet/JSP 執行於 Web 容器之中，EJB 執行於 EJB 容器之中，Java 應用程式客戶端執行於應用程式客戶端容器之中，不理解元件如何與容器互動，就無法真正善用或理解元件的行為。

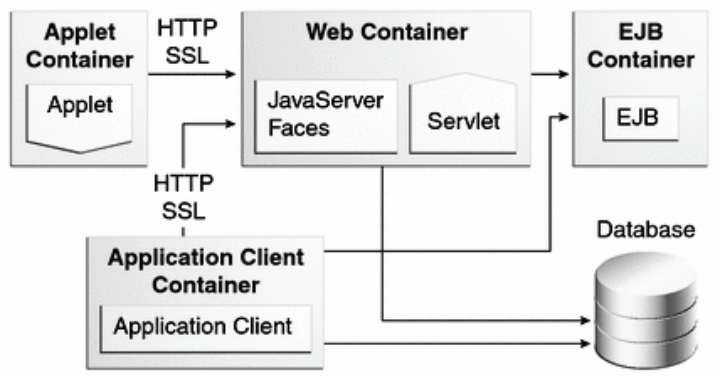

圖 1.6 摘自 http://download.oracle.com/javaee/6/tutorial/doc/bnacj.html

研究開放原始碼專案

Java 不僅是程式語言，也是個標準，在共同標準下有不同的實作方式，在 Java 領域的許多實作，都是以開放原始碼的方式存在，只要你有興趣，可以下載原始碼了解實作方式，**從中瞭解甚至吸收他人設計、實現產品的技巧或理念**。

許多基於 Java 各標準平台發展出來的產品也值得研究，例如測試框架（Framework）、Web 框架、永續（Persistance）框架、物件管理容器等，這些產品補足標準未涵蓋之處，各有其設計上優秀與精良之處，有些更進而回饋至 Java 而成為標準之一，重點是它們也多以開放原始碼的方式存在，讓開發人員可以使用、研究甚至參與改進。

提示 想要開始研究開放原始碼專案的使用與設計，JUnit 是個不錯的開始，可以參考我的線上文件：
http://caterpillar.onlyfun.net/Gossip/JUnit/

學習設計模式與重構

在程式設計上，「經驗」是最重要的，在經驗傳承上，歸納而言，無非就是「如何根據需求作出好的設計」、「如何因應需求變化調整現有程式架構」，對於前者，流傳下來的設計經驗就是**設計模式（Design pattern）**，對於後者，流傳下來的調整手法就是**重構（Refactor）**。

「如果我當初就這麼設計，現在就不會發生這個問題了！」這種對話應該很熟悉，「當初就這麼設計」就是所謂設計模式。「如果我當初先這麼改，再那麼改，就不會把程式改到爛了！」這種對話也經常聽到，「當初先這麼改，再那麼改」就是所謂重構。

無論是好的設計或不好的設計，都要有經驗傳承。經驗可以口耳相傳，可以從原始碼中觀摩，也可以從書籍或網路上優秀的技術文件中習得，對於初學者，從書籍或網路上優秀的技術文件學習設計模式與重構，是快速累積經驗的捷徑。

熟悉相關開發工具

除了累積足夠的實力與基礎，善用工具是必要的，開發工具可以避免繁瑣的指令、減少重複性的操作、提示可用的 API、自動產生程式碼、降低錯誤的發生，甚至執行各種自動化的測試、報告產生與發送郵件等任務。有些開發人員鄙視開發工具，這是不必要的，兩個實力相同的開發人員撰寫相同的應用程式，使用良好開發工具的人必然有較好的效率。

在 Java 的領域難能可貴的，是存在不少優秀的開發工具，而且多以開放架構、開放原始碼的方式存在，如 Eclipse IDE、NetBeans IDE 都是相當不錯的選擇，還可以搭配 Ant 建構工具、Maven 專案工具等一同使用，大大地提昇開發人員的產能。

建議初學 Java 的人，可以挑選一種開發工具來熟悉，所謂熟悉，不是指「下一步要按哪個按鈕、接下來要執行哪個選單」，而是指這些開發工具相關操作，是為了代勞你手動執行哪些指令，開發工具中的某些選項，是為了代勞你設定哪些變數，錯誤提示原本是來自 JDK 的什麼訊息等，以這樣的過程來熟悉開發工具，才能善用開發工具提昇產能，而不是受制於開發工具，如此就算換了另一套開發工具，也可以在最短時間內上手。

提示 ››› 本書會使用 NetBeans IDE，IDE 中的操作、設定與 JDK 指令的對照，會是本書的重點之一。