國立屏東大學 資訊工程學系 Java物件導向程式設計

1. Java語言簡介

Java是Sun公司在1995年所推出的物件導向程式語言。由於切合網際網路的需要，以及具有跨平台等優越性，Java在短短的幾年間，就躍升為主流的程式語言。本章將為您說明Java語言發展的緣由、相關的技術與應用等主題。以便在您開始學習之前，能夠Java能有多一點認識。

1.1 發展歷程

本節將就Java語言的發展背景與延革做一介紹。主要內容參考自Java Technology An Early History一文。

1.1.1 Green Team

Sun於1991年4月8日，由Patrick Naughton、Mike Sheridan與James Gosling組成了名為「Green Team」的工作團隊。Green Team主要的工作是，發展能在不同的作業平台上執行的分散式技術。其重點在於將發展出的技術，應用在消費性電子產品上。簡單來說，其目的是要發展出一種小型的裝置，該裝置上能以無線方式進行資料的傳輸，並且具有多媒體的處理能力。其實，他們的目標就是要開發類似今日平板電腦的裝置，不過在20多年前，那是非常先進的想法。

開發這樣一個裝置，牽涉到了硬體的設計、作業系統的配合以及系統開發工具等複雜的環節。Green Team並不是要自行開發所有的技術或工具，採用了「Hammer Technology」做為研發的策略，所以在很短的時間內就獲得了成果。

```
何謂Hammer Technology?
所謂的「Hammer Technology」指得是：將現有的各項技術組合起來，成為新的產品的方式。
採用此種專案開發時，不需要自行開發所需的技術或工具，而是在現有的技術或工具中，找尋適合者加以應用。
此種技術，強調的是系統整合的能力。
```

首先在硬體方面，他們直接採用了Sun當時已相當成熟的SPARC為基礎，稍加修改後，便有了一個相當穩固的硬體平台。而在作業系統方面，則是以UNIX為基礎，設計出符合該特定硬體的作業系統。至於系統開發工具方面，則是以C/C++語言為基礎。這樣一來，所有關鍵的技術都已齊備，新產品就很順利地誕生了。

1.1.2 Star Seven (*7)

在1992年9月3日，Green Team正式發表了其研發的成果，一台名為「Star Seven」的手持式產品。「Star Seven」也就是一個星號再加上一個數字7，可寫做『*7』。*7的發表，並不僅是硬體的部份，還包括了以UNIX相關技術為基礎所設計而成，名為「Green OS」的作業系統。此外，Green Team也設計了名為「Oak」的程式語言，以及Green OS的SDK(系統開發工具)。

為何叫做*7?
7這個名稱的由來，相當的有趣。因為當時在研發期間，Green Team是被隔絕於公司之外，一處安靜的場所中進行研發。同時，為了讓研發人員能專心，他們對外的聯繫唯一管道，就只有一個可撥回公司的電話而已。而該支電話，必須先按『』號，再按數字『7』，才能撥回公司。所以Green Team最後就以『*7』做為新產品的名稱。

*7在外觀上就如同今日我們所熟悉的PDA(個人數位助理)一樣，只不過還稍微地大了一點。它配備有一個5吋大小的彩色觸控螢幕，可顯示16位元的顏色。配合了友善的人機介面，相當易於操作。它採用PCMCIA的介面，並支援Flash Ram的檔案系統。更讓您激賞的是，它還支援無線的網路傳輸。從許多角度來看，*7甚至在今日都還可算是相當成熟的產品。

Duke的誕生
在*7的人機介面設計上，有一個虛擬的人物做為使用者與系統間的介面。您可以透過該個虛擬人物，來下達工作指令，或是查詢資訊。當時的這個虛擬人物，後來也成為了Java的識別圖示之一。這個虛擬的人物就是Duke。

圖1-1： 虛擬人物—Duke

1.1.3 James與OAK

由於Green Team成員中的James Gosling開發了一個新的程式語言—Oak，而Oak就是後來Java語言的前身。所以有不少人將Oak視為Java語言之父。事實上，James Gosling是一位相當活躍的資訊界前輩，在網路上也流傳了許多關於他的軼事，有興趣的讀者可以參考 James Gosling。

前面我們提到了*7的發展過程，其中使用了以整合為基礎的Hammer Technology，Green Team以SPARC的硬體架構為基礎，以UNIX為藍圖設計出Green OS，再搭配上以C/C++為參考，所設計出的全新程式語言—Oak，使得*7成為一項可算是相當成功的產品。本節將為您簡介這個後來成為Java語言的Oak語言，其設計的考量與特色。

為何叫做Oak?
Oak就是『橡樹』。之所以將這個程式語言命名為Oak，其原因也是相當的有趣。前面已經提過，在*7的研發期間，Green Team是被隔絕於公司之外，一處安靜的場所中進行研發。在這段形同閉關的日子中，研發人員稍微得以放鬆的就只有窗外的風景而已。而在他們的窗外看出去，就是一棵高聳的橡樹，所以這就是這個程式語言命名的由來了。

Oak的設計被要求要能滿足以下幾點：

- 小型
  - 由於新的程式語言必須要能在小型的手持式裝置上運行，所以它的執行環境必須要足夠小。舉例來說，如果在只有2MB記憶體的小型裝置上，您的執行環境卻需要1MB，那不就佔據了太多的系統資源了。
- 安全
  - *7的目標之一是要能在無線網路的環境中運行，所以在安全性的要求上就顯得特別重要，所以Oak也被要求要能確保程式執行的安全。常見的方式有，提供驗證合法軟體來源的能力，驗證程式碼的相容性等方法。

- 分散
  - 由於新的資訊系統的趨勢偏向於分散式運行環境，所以在*7設計時，當然也就將此點列入了重要的考量。以最簡單的方式來說明，所謂的分散式的概念，就是指一個完整的應用程式可以被分割成數個部份，並且可放置在不同的位置上，當執行時再動態地載入所需的部份加以執行。以另一個角度來看，也可將此概念延伸到多個應用程式共享相同的程式片段，也就是共享函式庫的觀念。
- 物件導向
  - 有別於結構化程式設計，物件導向(Object Oriented)具有資料封裝、繼承與多型等特性。在實際應用上，以物件導向方式所設計的程式，具有可重覆使用(Reuse)性以及易於維護等絕佳優勢，都使得物件導向成為新的趨勢。幾乎所有新的程式語言都具有物件導向的特性。
- 自動記憶體管理機制
  - 記憶體是電腦系統的重要部份，也是程式運行效能的關鍵。以往的程式語言，在記憶體使用方面，大多都需要由設計者自行配置、使用與回收記憶體區塊。若在設計上稍有不慎，記憶體的不當使用很容易造成系統的不穩定，或是更嚴重的當機情況的發生。Oak希望能具有自動記憶體的管理機制，使得程式設計的工作便得更為簡單(因為您不必再擔心記憶體管理的問題)，同時程式也更安全、穩定(因為相關的記憶體管理機制已經由程式語言本身提供)。
- 多執行緒
  - 現今的電腦系統具有多工處理能力。CPU(中央處理器)可以在不同的程式間切換，以形成同時(其實並不是真的同時)執行多個程式的效果。所謂的多執行緒(Thread)指得是在一個程式中，可切換執行的程式片段。如此不但系統可以同時執行多個程式，同一個程式中也可以同時執行不同的功能。例如一個讀寫檔案的程式，可以設計一個執行緒負責檔案的讀寫，然後再以另一個執行緒用來顯示讀寫進度的畫面。

1.1.4 FirstPerson

成功研發了『*7』後。Green Team於1992年10月1日，改制為獨立的FirstPerson公司。其主要目標是與有線電視的結合，曾經與時代華納公司洽談共同開發互動有線電視，可惜最後並未有具體的結果。

在開發有線電視市場失利後。FirstPerson公司便積極尋找可能的市場契機。在1993年時，隨著Mosaic的發表，全新的World Wide Web時代已揭開序幕。一時之間結合文字、影像等全新的網路資源存取，立刻吸引了無數人們的眼光。許多人，爭先恐後地投入WWW的懷抱。FirstPerson公司也相當看好這個市場，並且他們認為當時純粹以HTML來製作的網頁，還有很大的改善空間，例如即時性的互動多媒體內容還無法在網路上傳送、應用程式也不能在網路上傳送與執行。所以FirstPerson決定以推出一個更好的。WWW Browser。(Browser。瀏覽器)，來解決上述的問題。

1.1.5 WebRunner

FirstPerson公司的成果就是一個全新的瀏覽器，名為。WebRunner。。在當時(1995年)市場上已有了Mosaic與著名的Netscape這兩套瀏覽器。它們將。WebRunner。視為一個更好的瀏覽器，因為它能與Mosaic。Netscape相容，並且還提供了互動式的全新操作模式。這一切都是因為他們引入了一個新的元素—『可在網路上透過HTTP傳輸，將應用程式下載到瀏覽器上加以執行』。此一技術的背後，就是以Oak為基礎所設計的Java技術。

為何叫做Java?
以Oak語言為前身的Java語言，為何要命名為Java？其實當初最優先的名稱就是維持Oak的名稱，無奈該名稱已被註冊。其後選用的一些名字，如Café。Coffee等，也都遇到同樣的問題。最後只好在許多備

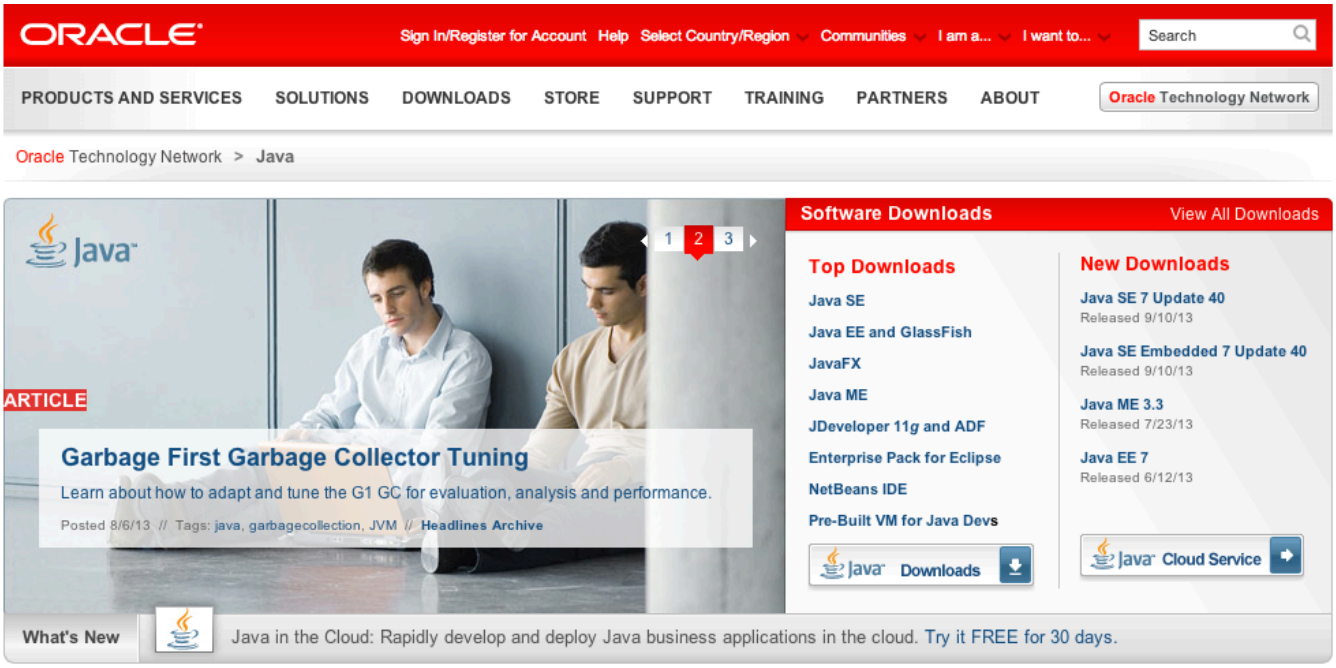
用的名字當中，選擇了當時尚未被註冊的Java做為新的名稱。

1.1.6 www.java.com

當初推出WebRunner時，所內建的Java是1.0a版。隨後WebRunner被更名為HotJava並且Sun公司推出了http://java.sun.com網站，成為Java技術的官方入口網站。在1995年5月23日Java的JDK(開發工具套件) 1.0a2開始在java.sun.com上免費供人下載，在短時間內就獲得許多領導廠商的承諾，願意共同推動Java成為開放式的標準。

2009年4月Oracle公司併購了Sun所以Java相關技術也就併入了Oracle公司旗下。Java官方網站(圖1-2)已經成為所有Java開發人員，最常拜訪的網站之一了。不論您需要的是什麼(下載開發工具、疑難排除、瞭解最新的資訊、甚至是想找份與Java有關的工作)，都可以在此找到答案!

圖1-2：http://www.oracle.com/technetwork/java



1.2 Java的特性

本節將為您簡單介紹Java語言的特性。其中有一些好的特性是承襲自結構化語言，有的則是源自於物件導向語言。當然Java語言也有其獨創的功能，使它能成為廣受歡迎的語言。以下是其特性的介紹：

1.2.1 物件導向(Object-Oriented)

您應該已經知道Java是一種物件導向的程式語言。物件導向有許多優於結構化程式語言之處，例如資料的抽象與封裝性、物件的繼承性等。

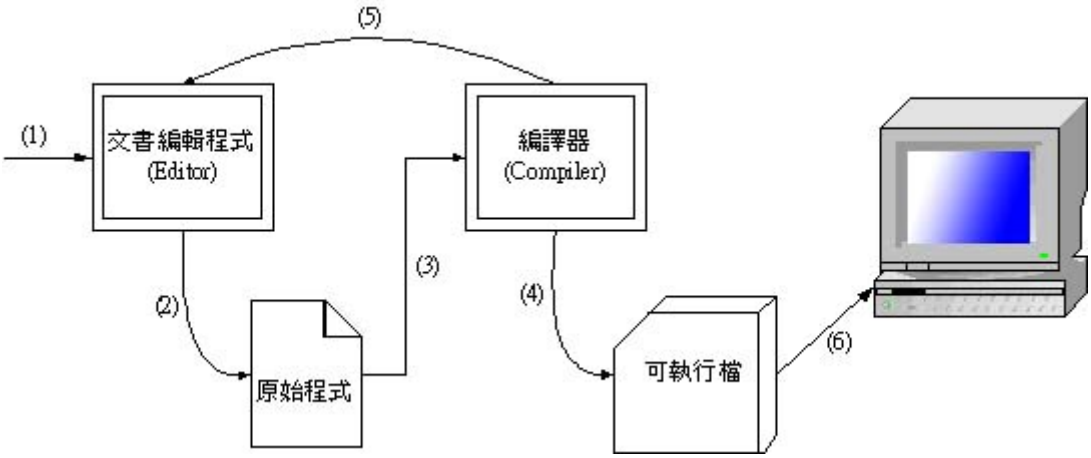
早期結構化語言的代表，如Pascal與C語言，隨著物件導向的風潮，也都陸續被Object Pascal與C++等語言所取代Java語言做為一個新的語言，當然也不能例外地是物件導向的程式語言；同時以筆者個人的觀點來看Java的設計上可說是十分吻合物件導向的精神(至少比那些宣稱符合物件導向精神的程式語言好的太多了)。關於物件導向，本書第<???????>章，會有深入的說明。

1.2.2 跨平台

一個程式從開始撰寫到可以執行，中間會經過許多的步驟。通常的情況是，先使用文書編輯軟體或開發工具，進行原始程式的編寫；然後透過編譯器(Compiler)將原始程式進行編譯(也就是將原始程式轉換成機器語言)。經過上述過程，就產生了可在電腦上執行的程式。請參考圖1-3。

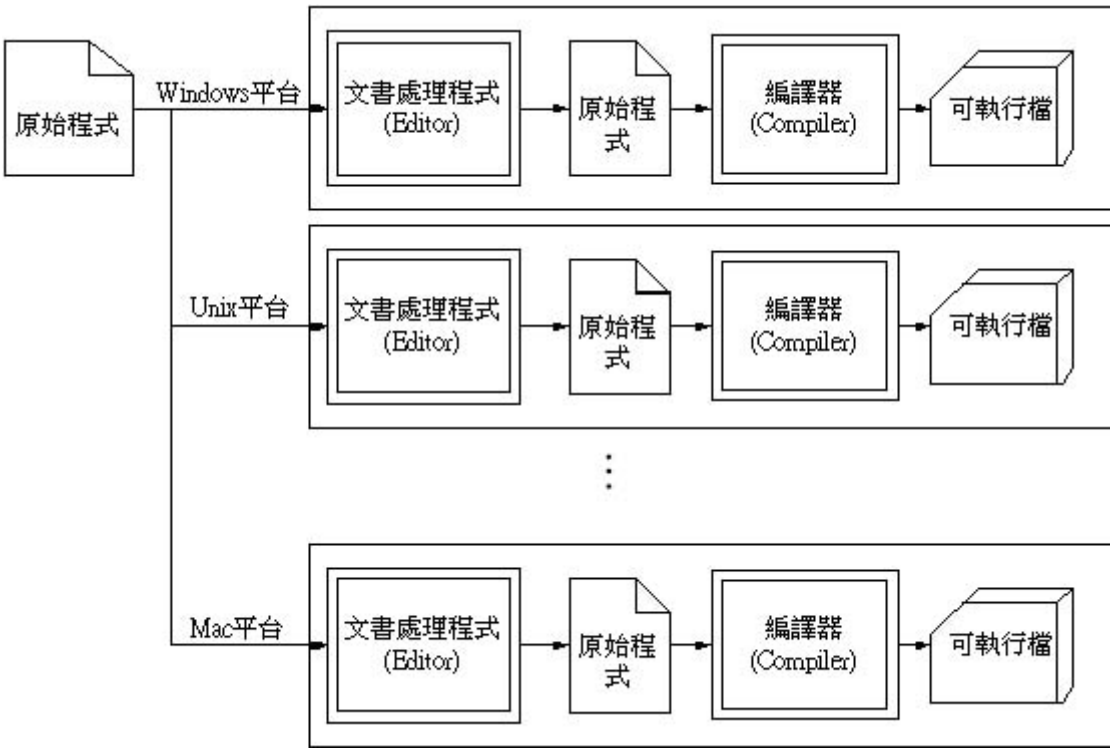
1. 以文書處理器(或是整合型的開發工具)撰寫原始程式。
2. 完成後，產生原始程式檔。
3. 將原始程式交由編譯器(Compiler)編譯。
4. 編譯後，會產生符合作業平台的可執行檔(例如Windows平台下的EXE檔)。
5. 若是編譯時發生錯誤，則重新回到步驟1進行除錯。
6. 最後就可在作業平台上執行程式。

圖1-3：傳統程式開發流程圖。



上述的開發流程，對於一些可能會在不同平台(通常指的是不同的硬體或作業系統)上執行的程式來說，這就會比較另人討厭了。如果一個程式，必須要在一個以上的平台上面執行的話，這種程式就稱為跨平台(Cross Platform)程式。因為在不同平台上的原始程式都需要經過修改，並且在不同平台上重新編譯後才能執行。如果原本所採用的開發工具、程式語言，在新的平台上並沒有對應的工具或語言，那麼就可能需要重新加以撰寫了。圖1-4就是這種情況的一個典型的例子。

圖1-4：傳統跨平台程式開發流程圖。

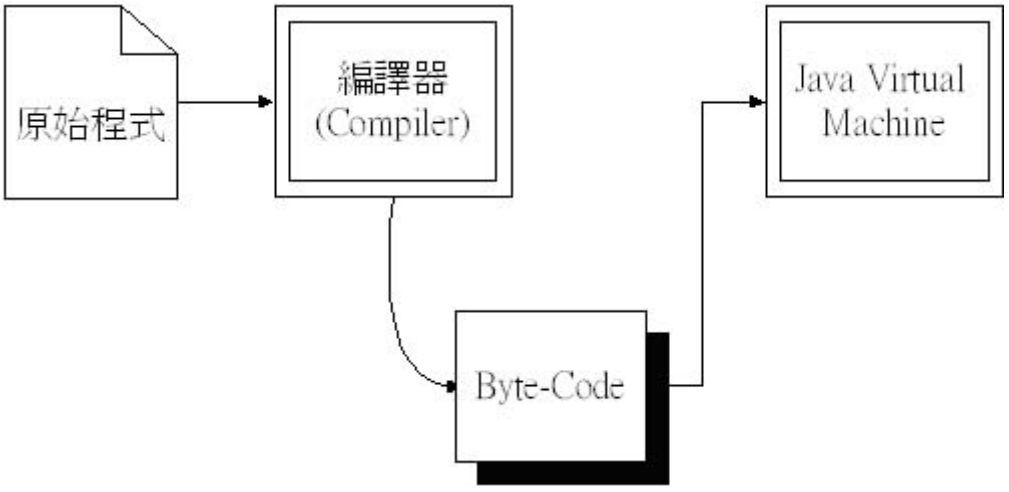


在圖1-4中，原本寫好的程式要拿到不同的平台上執行時，必須先各別在不同的平台上進行修改與編譯的動作，然後才能得到在該平台上的可執行檔。這樣的過程不但麻煩，同時也有肇因於不同平台的不同功能，使得同一個程式在不同平台上的功能有不竟相同的現象。關於此問題，由於Java在實作上採用了新的設計，因此順利地解決了跨平台的問題。以下我們將為您說明相關的細節。

1.2.2.1 虛擬機器

使用Java所設計的程式，並不能直接在作業平台上執行；而是要透過一個建立在各個平台之上的軟體來加以執行。圖1-5簡單地為您說明了這個概念。

圖1-5：虛擬機器示意圖。



在圖1-5中，一個以Java所撰寫的程式，會先透過編譯器(Compiler)轉譯成Byte-Code□所謂的Byte-Code並不能直接執行，而是要透過一個名為Java Virtual Machine(Java虛擬機器，簡稱JVM)的軟體才能加以執行。

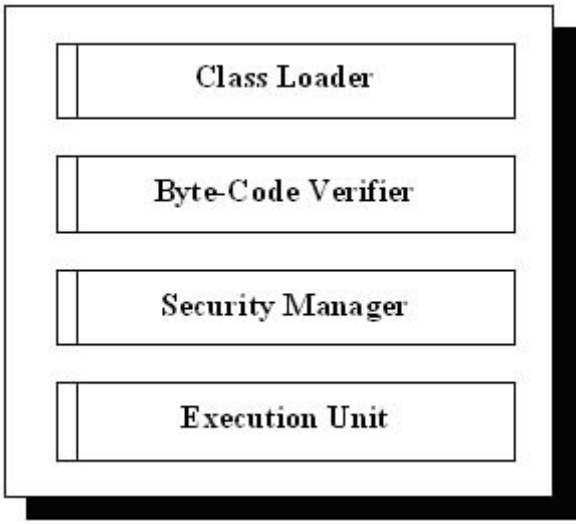
您可以將JVM視為一個軟體，它接受Byte Code的格式(也就是以Java所撰寫的程式，並經由編譯器編譯過所產生的格式)，然後將Byte Code中所要執行的操作加以執行。所謂的Byte-Code與機器語言十分類似，唯一的差別是它只能在虛擬的JVM上執行而已。

假設原本的程式中有一行print的指令，用來將「Hello!」印在螢幕上。編譯器就會將這行程式碼，轉變成JVM可以閱讀的格式。然後在執行時，是先啟動JVM，再由JVM來完成這個印出字串的動作。

JVM內部包含了以下四個部份：

1. Class Loader
   - 類別載入器。由於以Java所撰寫的程式，都是以類別的形式存在(關於此點在本書後續的內容中，將會有詳細的介紹)。所以JVM中首先要由這個類別載入器，將Java所設計的類別載入(當然，所載入的是那些已成為Byte-Code格式的類別檔案)。
2. Byte-Code Verifier
   - Byte-Code驗證器：JVM透過類別載入器載入類別後，接著要進行的就是檢查所載入的類別是否符合Byte-Code的格式。這個動作是由Byte-Code驗證器所負責，其目的是避免因為執行格式不正確的程式，而導致程式的錯誤。
3. Security Manager
   - 安全管理。所載入的Byte-Code經驗證後，接著就會進行安全性的檢查，以避免執行到惡意的程式，導致執行平台的資料外洩或遭到破壞。
4. Execution Unit
   - 執行單元。經過重重檢查之後，程式就會由執行單元加以執行。

圖1-6：Java虛擬機器內部機制。



1.2.2.2 Write once, Run anywhere!

目前幾乎所有的平台上，都有專屬的Java虛擬機器。凡是使用Java所撰寫的程式，不論在那一個平台上的虛擬機器，都可以得到一樣地執行效果。換句話說，所有使用Java開發的程式，都可以享有「Write Once，Run Anywhere!(寫一次，到處都可以執行)」的好處。

由於不同平台間的差異頗大，所以事先針對不同的平台所撰寫的虛擬機器，其目的並不是在於要讓特定平台上的JVM能發揮最大的功效；相反地，JVM可能僅能使用到原本作業平台的部份功能。因為不論作業平台有多少功能，JVM的目的是要在不同平台間，建立一個獨立的虛擬平台。對所有的Java程式來說，不論在哪一個平台上的JVM執行，其結果都是一樣的。所以JVM的設計參考了大多數平台所共同擁有的功能、架構，所設計出的結果就有可能使得JVM僅具有原本平台上的部份功能。但為了確保不同平台上的JVM能提供一樣的功能，這點是不得不為的措施。此外，JVM在設計上，也採用了許多技巧，使得不同平台上的JVM都能有一樣的功能。舉例來說，Java語言提供32位元的整數，在那些無法提供32位元整數運算的平台上，JVM就可能以兩個16位元的整數來模擬一個32位元的整數。

JVM除了出現在不同的平台外，在網際網路的瀏覽器上也嵌有JVM，我們可以透過網際網路，在瀏覽器上來執行Java的程式。這點也是讓Java廣受歡迎的原因之一。

1.2.3 簡單

Java的設計有兩項很重要的考量因素：簡單學習、簡單使用。所以Java被設計成與C/C++類似，但卻沒有刪除了一些C/C++中，容易讓人困惑或是難以理解的部份。例如Java並沒有和C/C++相同的struct、union，同時Java也沒有goto等敘述。除了上述幾點外，以下兩點是Java語言之所以簡單的主要理由。

- 沒有指標
  - 指標(Pointer)是在程式語言中，用以直接存取記憶體位置的方式。對於程式設計者來說，指標的使用可以提昇效率；但也可能因為錯誤的使用導致非預期的情況發生，嚴重者還可能當機。在Java語言中，由於不提供指標的使用，所以在設計上就簡單的多了。
- 自動記憶體管理
  - 除了沒有指標外，許多程式語言(包含C/C++)在使用記憶體空間前，必須先要求配置適當的空間，並且在使用完成後將空間釋放掉。如果忘記釋放，則會造成可用空間的不足等問題，Java採用了Garbage Collection(記憶體資源回收)的方式，自動幫我們處理這些問題。所以程式設計的工作又更加簡單了。

1.2.4 動態與分散

本節將以最簡單的方式，為您說明什麼是Java語言的動態與分散特性。

- 動態
  - 一個完整的Java程式可由多個類別組合而成。當程式執行時，可視情況分別載入不同的類別，此點就稱為動態。相對的，所謂的靜態則是必須要一次載入所有的類別，才能加以執行。
- 分散
  - 前述的動態載入特性，是指程式視執行所需動態地載入不同的類別。而分散則是指，所需要載入的類別可以散落在網路上的不同地方。當需要時，再透過網路加以下載。

當然，一些常用的類別，就可以事先寫好一份放在網路上，供多個程式使用。

1.2.5 網際網路能力

Java語言最吸引人注意的就是其網際網路的運作能力。這方面除了有完整的Socket API外，不論在Web的Client端或Server端，Java都有適當的技術可供運作(例如Cleint端的Applet、Server端的Servlet等)。詳細的內容請參考後續的說明。

1.3 Java技術應用

以下將為您簡單介紹不同的Java技術及其應用：

1.3.1 Applet

Applet是『小型應用程式』的意思。以Java語言所撰寫的Applet，是放置於Web Server上，當使用者要執

行時，先透過瀏覽器連接上Server端，並且下載Applet的類別檔案(這些動作是自動完成的)到使用者的電腦上，最後再由瀏覽器上內嵌的JVM加以執行。

因為Applet是在Web架構下，讓Client端的瀏覽器能擁有多媒體、即時、互動等特性的技術，所以被廣泛地使用在許許多多的網站上。

但也由於Applet是在網路上傳送、執行，所以在檔案大小與安全性等方面要求格外嚴格。在Java的開發工具中，已包含Applet類別，我們只要繼承該類別加以設計新的功能，便能很快地完成小而美的Applet。至於安全性方面，Java也限制了一些操作是不能在Applet中進行的(例如：I/O的存取)，以避免執行了來自網路上，具危害性的Applet。

1.3.2 Application

Application就是指一般的應用程式。您當然可以像使用C/C++等語言一樣，以Java來設計一般的應用程式。目前已有許多廠商看好Java的跨平台等特性，積極投入開發Java的應用程式。舉凡商務應用、教育研究甚至是電玩遊戲等領域，都已經有了Java的產品。

1.3.3 JavaScript

JavaScript是另一種可應用在網頁設計的程式語言。許多人都以為它是Java技術的分支之一，但它並不是！它與Java語言並沒有什麼太大的關係！只是名稱上有些類似而已。

它是由Netscape公司所開發的LiveScript所衍生而來的，是寫在網頁上並可在瀏覽器上執行的Scripting語言。目前在主要的瀏覽器(Netscape與Microsoft的IE)中，分別支援了JavaScript與相容的Jscript。

基本上JavaScript的程式碼是內嵌於HTML的檔案中的，當瀏覽器將Server上的HTML檔案載入後，再由內建於瀏覽器的JavaScript(或JScript)的解譯器，來將程式碼加以執行。類似的技術還有VBScript、PerlScript等。

1.3.4 Servlet

Servlet是Java與Web架構結合的好例子。所謂的Servlet是由Java所撰寫的程式，經編譯成Byte-Code後，放置於Web Server上。當有使用者從網路連結到此Servlet時，支援Servlet的Web Server就會將類別檔案載入，並且加以執行，最後將結果傳回給使用者的瀏覽器。其後，如果有其他的使用者再次連結此一Servlet時，就不需要再次載入，可以直接執行。

這種方式的執行效率或安全性等，都比其它相關的Web技術好上許多，也是目前重要的網站相關技術之一。

1.3.5 Java Server Page(JSP)

Java Server Page又可簡稱為JSP。同樣是內嵌在網頁中的程式語言，不過它是在Server端執行的。避免了JavaScript等程式，因為要在瀏覽器上執行，所以自然可以在瀏覽器上看到原始程式的缺點。

此外，JSP的執行會將程式碼，轉成Servlet形式的Byte-Code，然後再以Servlet的方式來運作。因此，除了初次啟動時須等待一段時間外，其執行效率也相當地好。

1.3.6 其它

事實上，與Java相關的技術絕不只本章所介紹的幾項而已。包含EJB(Enterprise Java Beans)、JINI、JMS(Java Message Service)、JTA(Java Transaction)、JTS(Java Transaction Service)等，現在有太多、難以計數的Java相關技術存在。本書也不可能將所有的技術都加以介紹，不過希望本書能做為初學Java的一個好的指引。

1.4 JDK與JRE

在Java的開發與執行上，JDK與JRE無疑扮演了重要的角色，JDK是Java Development Kit的縮寫，JRE則是Java Runtime Environment的縮寫。以下我們將為您簡單地介紹它們的用途。

- Java Development Kit (JDK)
  - Java Development Kit也就是所謂的Java開發工具集，它是最基本的開發工具。在JDK中，包含了編譯器、偵錯工具、輔助說明文件產生器等，各種在開發階段所需要的工具。
- Java Runtime Environment (JRE)
  - JRE也包含了一些工具，而這些工具存在的目的是要幫助我們來執行Java的程式。例如JVM就是JRE中重要的一部份。

不論是JDK或是JRE，您都可以在http://www.oracle.com/technetwork/java上取得最新的版本。在過去尚未有足夠好的開發工具時(筆者指的是像Netbeans這種絕佳的開發工具)，我們都是直接使用JDK來進行開發的工作，雖然有時比較麻煩，但卻是近距離觀察Java的好方法。如果有興趣的話，建議您也可以說一點時間去瞭解更多關於JDK的細節。