本章重點：

★ Java 直譯器
★ 策略檔案
★ 類別路徑
★ Java 編譯器
★ JAR

3

基本工具

從傳統的命令列環境，到 IDE 環境，像是 WebGain 的 Visual Cafe、Borland 的 JBuilder、Tek-Tools 的 KAWA、Sun 的 Forte for Java，你有很多不同的 Java 開發環境可以選擇。本書的範例是使用 Solaris 和視窗版的 SDK，所以我們只會介紹這些工具。當我們提到直譯器（interpreter）或是編譯器（compiler）時，我們都是指這些工具的命令列（command-line）版本，所以這本書是稍微偏向 Unix 或是類似 DOS 模式的環境，但是我們這?敘述的 Sun 的 Java 直譯器和編譯器的功能應該和其他環境一樣才對。

在本章中，我們會說明你需要編譯和執行 Java 程式的相關工具。本章的最後部分則是告訴您如何將 Java 的類別檔案壓縮到 JAR 檔案中，對於擁有你信任簽章（signature）的類別如何給予較大的基本權力。

Java 直譯器

Java 直譯器（Java interpreter）就是執行 Java 虛擬機器和執行 Java 應用程式的軟體，它可能是一個獨立程式，如：SDK 的 java 程式，或是大型程式（如：Netscape Navigator 網頁瀏覽器）的一部份。直譯器本身應該是用針對平台的原生編譯語言所寫的，但是像編譯器、開發環境等其他工具就可以用 Java 去寫了（而且應該是，因為為了使 Java 開發環境有最大的可移植性）。Sun 的 Forte for Java 就是純 Java IDE 的例子。

Java 直譯器執行 Java 執行時期系統的所有動作，它可以載入類別檔和執行編譯過的位元碼，並檢驗從不被信任的來源所載入的類別。在支援動態或及時編譯的程式中，直譯器也可以是將 Java 位元碼轉換成原生機器指令的特殊編譯器。

在本書的其餘部分，我們會寫獨立的應用程式和 applet，兩者都是由直譯器執行的 Java 程式，不同的是一個獨立的應用程式是可以獨立執行的完整程式，而 applet 比較像是嵌入式的程式模組。直譯器無法直接執行 applet，因為它只是被當成大程式的一部份來使用，要執行 applet，你可以使用如 Netscape Navigator、Sun HotJava 之類的瀏覽器，或是用 SDK 內附的 appletviewer 工具。HotJava 和 appletviewer 都是可直接由 Java 直譯器執行的獨立應用程式，實作了執行 Java applet 所需要的其他結構。

Sun 的直譯器就叫 java。在獨立的 Java 程式中，每個類別包含一個 main() 方法，?面有一開始就被執行的敘述。要執行某個應用程式要先執行直譯器，且指定這個應用程式類別作為參數。你也可以指定直譯器的選項以及是要傳遞給程式的參數：

```
% java 直譯器選項 類別名稱 程式的參數
```

其中的*類別名稱*應該要用完整定義名稱，如果有套件的話，也必須要包含套件名稱。但是要注意，你不必加上 .class 附檔名。看看底下的例子：

```
% java animals.birds.BigBird
% java test
```

直譯器會搜尋類別路徑?的類別，類別路徑（class path）是儲存類別套件的目錄列表。我們在下一節會詳細地討論類別路徑。類別路徑是由一個環境變數來設定，你也可以用命令列選項 -classpath 來重新設定。

在載入命令列所指定的類別後，直譯器就會執行類別中的 main() 方法，從此時起，程式可以開始額外的執行緒、參考其他類別、建立使用者介面或其他結構，如圖 3-1 所示。

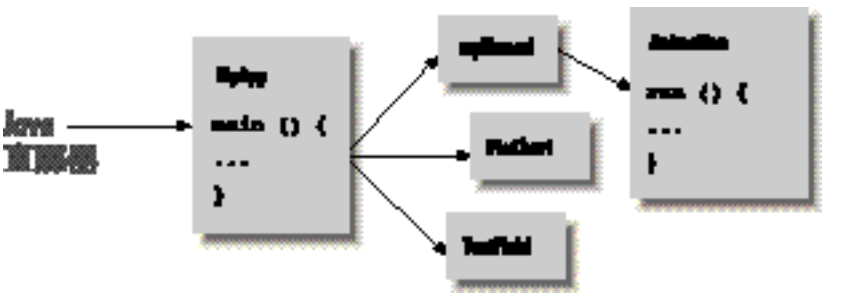

圖 3-1. 開始執行 Java 程式

main() 方法必須要有正確的方法特徵。方法特徵（method signature）是有關這個方法資訊的集合，像是 C 語言的原型（prototype）、或是在其它語言的轉送函式宣告。方法特徵包含方法的名稱、型態、可見度（visibility）、參數、以及回傳值型態。main() 必須是 public、static，並以 String 物件的陣列作為參數，且沒有回傳值（void）：

```
public static void main ( String [] myArgs )
```

因為 main() 是公開、靜態的方法，所以它可以被其他的類別直接存取，只要其他類別使用包含這個 main() 的類別的名稱。我們會在第四章到第六章討論可見度修飾字 public 的含意和 static 的意義。

main() 方法的唯一參數，就是 String 物件的陣列，它存放傳遞給程式的命令列參數。和 C 語言一樣，參數的命名不重要，重要的是參數的型態。和 C 不一樣的是，myArgs 的內容是真實陣列，所以不需要額外的計數參數，因為 myArgs 知道它包含多少個參數並且可以恰當地提供需要的訊息。

```
int argc = myArgs.length;
```

Java 和 C 不一樣的另一點是：Java 的 myArgs[0] 是指命令列的第一個參數，而不是程式的名稱。如果你已經習慣 C 的命令列參數順序，那你對於這點差異必須特別的小心。

Java 直譯器會繼續執行，直到初始類別的 `main()` 方法結束，且所有的執行緒已經完成。當程式其餘部分也完成時，像特殊設計的「幽靈」執行緒也會無聲無息的結束。

策略檔案

Java 2 提供了一個簡單的機制來保護你的電腦免於被病毒之類的程式惡意破壞。如果你從網路上的某處下載了一個程式，你要如何防止它竊取你電腦上的資料，並且再把它丟回到網路上？你要如何防止癱瘓你電腦或是刪除你磁碟上檔案的惡意程式？大部分的電腦平台對這些問題都無解。

Java 2 提供有力的方式來限制執行程式的動作。在 Java 2 以前，許多對安全性的要求都是和 applet 有關，在安全性限制下執行的 applet 可以避免它執行某些危險的動作，像是對你的磁碟做讀寫動作、或是連接到網路上的任一台電腦。Java 2 對應用程式也簡單地採用類似 applet 方式的安全性限制，此外，對你允許的程式做存取限制的細微調整也是很容易的。例如，你可以只允許程式存取磁碟上的某個目錄，或是只允許存取網路上的某些特定位址。

這為什麼很重要呢？假設你需要某種程式，像是行事曆或是通訊錄管理員好了，你到你喜愛的搜索引擎尋找，並且找到一個正如你需要且看似強大的 Java 程式。你下載並執行這個程式，但是極有可能你下載的並非是你想的，它可能是感染你電腦的病毒，或是刪除你磁碟上檔案的搞怪程式。這種情形下，限制程式的行為就是一個很棒的措施了。

預設的安全性管理員

你可以使用直譯器的選項參數來安裝預設的安全性管理員，安全性管理員會強制執行許多規則，就像對 applet 那樣。為了要看看它是怎麼做的，我們先寫一個惡搞的小程式，它會自己連到網路上的某些電腦。（在第十一章和第十二章我們會討論網路程式設計的細節。）

```
// 檔案: EvilEmpire.java
import java.net.*;

public class EvilEmpire {
    public static void main(String[] args) throws Exception{
        try {
            Socket s = new Socket("207.46.131.13", 80);
            System.out.println("Connected!");
        }
        catch (SecurityException e) {
            System.out.println("SecurityException: could not connect.");
        }
    }
}
```

如果你用直譯器執行這個程式，它會連到網路上：

```
C:\> java EvilEmpire
Connected!

C:\>
```

這很令人害怕。我們安裝預設的安全性管理員，如下：

```
C:\> java -Djava.security.manager EvilEmpire
SecurityException: could not connect.

C:\>
```

這樣就好多了。但假設這個程式真的有個合理的原因需要去連接網路，為了安全起見，我們希望在某個適當的地方可以不使用預設安全性管理員，但是必須經由我們的許可（permission），程式才可以執行網路連線。

policytool 公用工具

若要允許我們的 EvilEmpire 做網路連線，我們要先建立一個包含許可的策略檔案（policy file）。SDK 1.2 和之後的版本中，有個便利的工具叫做 policytool，可以幫你建立策略檔案。從命令列啟動 policytool：

```
C:\> policytool
```

當 policytool 啟動卻沒有找到任何預設的策略檔案時，你會有一個錯誤訊息，但不必擔心，直接按下 OK，忽略這個訊息。

我們想為 EvilEmpire 程式加入一個網路許可。程式是經由它的來源做確認，這個來源稱為 codebase。codebase 是一個 URL，在這個例子中是 file:，一個指到在你磁碟上 EvilEmpire 程式位置的 URL。

啟動 policytool 後，你應該可以看到它的主視窗，如圖 3-2。按下 Add Policy Entry 後另一個視窗會跳出，如圖 3-3 但是欄位是空白的。

首先，如圖 3-3 所示在 CodeBase 中填入包含 EvilEmpire 目錄的 URL，然後按下 Add Permission。另一個視窗會出現，如圖 3-4。

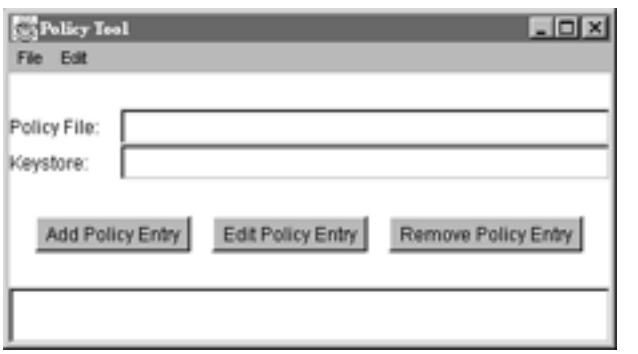

圖 3-2. policytool 視窗

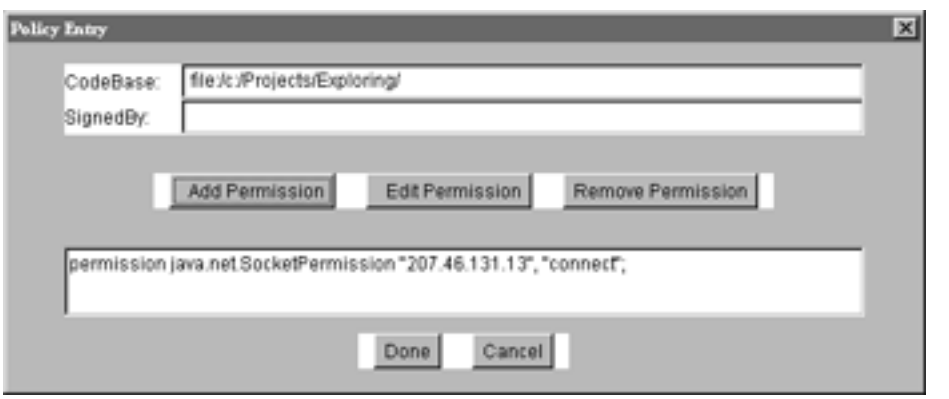

圖 3-3. 新增一個策略項目

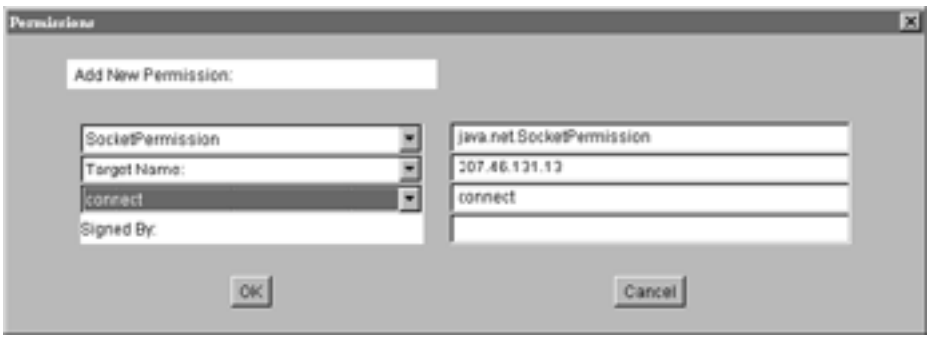

圖 3-4. 建立新的許可

從第一個下拉式選單選擇 SocketPermission，然後在右邊的文字欄位填入 EvilEmpire 要連結的網路位址。最後從第三個下拉式選單選擇 connect，按下 OK。你應該可以在策略項目視窗中看到新的許可，如圖 3-3 所示。

按下 Done 完成新策略的建立，然後從 File 選單中選擇 Save As 將策略檔案存成一個容易記的名字，如 EvilEmpire.policy。你現在可以結束 policytool 了，我們已經完成了。

你剛剛建立的策略檔案並沒有什麼神奇的地方，用文字編輯器看看這個檔案，它只有一個簡單的語法。以下是是我們剛建立的策略檔案最重要的部分：

```
grant codeBase "file:/c:/Projects/Exploring/" {
    permission java.net.SocketPermission "207.46.131.13", "connect";
};
```

你也可以不使用 policytool，只要用文字編輯器就可以產生策略檔案了，如果這樣你覺得比較習慣的話。

使用策略檔案

我們已經知道如何建立策略檔案了，現在接著要使用這個檔案。你可以用直譯器的命令列選項告訴預設的安全性管理員該使用哪個策略檔案：

```
C:\> java -Djava.security.manager -Djava.security.policy=EvilEmpire.policy
EvilEmpire
Connected!
```

我們已經用策略檔案給予網路連線的許可，所以現在 EvilEmpire 可以做網路連線的動作。預設安全性管理員在其他方面依然有保護功用，所以除了 EvilEmpire 自己所在的目錄外，它不可以讀寫磁碟上的檔案；除了指定的連線外，它也不行連到其他的電腦。現在花點時間感受一下這種溫暖昏炫的成就感吧。

到了第二十章，當我們解釋已簽章的 applet 時，你會再度看到 policytool。在本章中，codebase 是由 URL 來檢驗的，但這不是最安全的方式，經由高明的網路惡作劇，一個厲害的仿冒者可以欺騙你它的程式來源。使用已加密簽章的 applet 比較值得信賴，在第二十章有完整的細節說明。

類別路徑

對在 DOS 或 Unix 平台工作過的任何人來說，路徑（path）的觀念應該很不陌生，這是提供程式去某些地方找尋資源的一個環境變數，最常見的例子就是可執行程式的路徑。在 Unix 介面中，環境變數 PATH 是用冒號隔開的搜尋目錄列表。Java 的 CLASSPATH 環境變數也類似這樣，它是包含類別檔案的套件所在位置的搜尋列表。不管在電腦上是要尋找套件或類別，Java 的直譯器和編譯器都是使用 CLASSPATH 這個變數。

類別路徑可以是個目錄名稱或是類別的壓縮檔（class archive file）。對於類別壓縮檔案，Java 支援它自己的 Java 資料壓縮格式（JAR）和常用的 ZIP 格式。JAR 和 ZIP 實際上都是相同的格式，只不過 JAR 壓縮檔包含了額外的檔案，用以描述每個壓縮檔的內容。JAR 檔案可以用 SDK 所附的工具 jar 來產生，其他產生 ZIP 的壓縮工具也到處可見。使用壓縮格式可以將大群類別檔集中在一個檔案內加以發送，如果需要，Java 直譯器可以自動從壓縮檔中只取出個別的類別檔案。

設定這些類別路徑的方式和格式會隨著系統而異，在 Unix 系統上，你可以用冒號（:）分隔目錄和類別壓縮檔案的列表：

```
CLASSPATH=/home/vicky/Java/classes:/home/josh/oldstuff/foo.zip:.
```

在微軟視窗系統中，CLASSPATH 環境變數是用分號（;）分隔目錄和類別壓縮檔案的列表：

```
set CLASSPATH=D:\users\vicky\Java\classes;.
```

在第一個 Unix 的例子中，類別路徑指定了三個位置：一個是使用者的目錄，一個是在其他使用者的目錄內的 ZIP 檔，一個是目前的使用目錄（.）。類別路徑的最後一個部分，也就是目前的使用目錄，在對類別還不熟悉的時候是很有用的，但通常將目前使用的目錄放進任何路徑中都是不好的習慣。

Java 直譯器和其他的命令列工具也知道如何找到核心類別，這些類別在 Java 安裝完就會存在了，例如，在 java.lang、java.io、java.net、javax.swing 套件中的類別都是屬於核心類別。在你的類別路徑中不需要包含這些類別，直譯器和其他工具就可以自己找到它們了。

要找其他的類別，直譯器會依序尋找在類別路徑中的位置，它會尋找結合路徑位置和類別完整定義名稱的位置。例如，要尋找 animals.birds.BigBird 類別，搜尋的類別路徑目錄是 /usr/lib/java，則直譯器會到 /usr/lib/java/animals/birds/BigBird.class 尋找；若要在類別路徑上的 ZIP 或 JAR 壓縮檔中尋找，例如 /home/vicky/Java/utils.classutils.jar，則直譯器會在這個壓縮檔中尋找檔案 animals/birds/BigBird.class。

對直譯器 java 和編譯器 javac 來說，類別路徑也可以用 -classpath 選項來指定：

```
% javac -classpath /pkg/sdk/lib/classes.zip:/home/pat/java:. Foo.java
```

如果你沒有指定環境變數 CLASSPATH，它會預設是目前的目錄（.），這表示在你目前工作目錄中的檔案永遠都是可以使用的。如果你改變類別路徑，但沒有把目前工作目錄包含進去，那麼這些檔案就不行使用了。

Java 編譯器

在本節中，我們要介紹 javac，它是 Sun 的 SDK 所提供的 Java 編譯器；如果你恰巧使用其他的發展環境，你可以直接跳到下一節去。編譯器 javac 是完全用 Java 寫的，所以它對任何平台都支援 Java 執行時期系統。在語言發展中，支援自己的發展環境是很重要的一個階段，在移植直譯器時，Java 只要提供現成的編譯器就可以輕易達成了。

javac 將原始碼轉編譯成一個包含 Java 虛擬機器位元碼的類別。依照慣例，原始檔名的副檔名用“.java”，編譯後的類別副檔名用“.class”。每個原始碼檔案都是一個單一的編譯單元，你在第六章會知道，在每個編譯單元內的類別擁有相同的特徵，像是 package、import 敘述等。

javac 允許每個檔案有一個公開的類別，並且規定檔名必須和公開類別的名稱一致，如果不一致的話，javac 就會產生一個編譯錯誤訊息。一個檔案可以有多個類別，其中只有一個類別是公開的。應避免將許多個類別壓縮到同一個原始資料檔中，在一個 .java 檔案中包含那些非公開的類別，是將這些類別和公開類別緊密連結的一個簡易做法；但你也可以考慮使用內部類別（參考第六章）。

舉例來說，將下列程式放到 BigBird.java 中：

```
package animals.birds;

public class BigBird extends Bird {
    ...
}
```

然後編譯它：

```
% javac BigBird.java
```

不像直譯器只需要一個類別名稱作為參數，javac 需要一個檔名。例子中的命令會產生類別檔 BigBird.class，放在原始檔所在的目錄內。當測試簡單範例程式的時候，把類別檔和原始檔放在同一個目錄很有用，但對大部分真實的應用程式來說，你需要將類別檔放在類別路徑中適當的位置。

你可以用 -d 選項對 javac 指定所產生的類別檔要存放的目錄，指定的目錄會被當成類別階層的根目錄，所以 .class 檔案會被存放在這個目錄或是它的子目錄（依據這個類別是否被放在一個套件中來決定）。如果需要的話，編譯器會自動產生所有的子目錄。例如，我們使用下列命令可在 /home/vicky/Java/classes/animals/birds/BigBird.class 中建立 BigBird.class 檔案：

```
% javac -d /home/vicky/Java/classes BigBird.java
```

你可以在 javac 命令列中指定多個 .java 檔案，編譯器會為每個原始檔產生一個類別檔。但如果你就的類別將要參考到的其他類別已經編譯過了，就不必將被參考的類別列出來。在編譯過程中，Java 會使用類別路徑去解析其他的參考類別。如果我們的類別參考到 animals.birds 的其他類別或其他套件的類別，在編譯時就應該把對應的路徑加入類別路徑中，這樣 javac 才可以找到對應的類別檔案。

Java 的編譯器比一般的編譯器還要聰明，取代了 make 的一些功能。例如，javac 會對所有被參考類別比較原始檔和類別檔的修改時間，且如果需要的話會重新編譯。只要原始檔和類別檔是在同一個目錄中，編譯後的類別就會記得它編譯時的位置，所以如果需要時就可以重新編譯原始檔一次。在之前的例子中，如果類別 BigBird 參考到另一個類別 animals.furry.Grover，javac 會在套件 animals.furry 中尋找原始檔案 Grover.java，且如果需要的話會重新編譯，將 Grover.class 檔案更新。

雖然預設中 javac 只會檢查被其他原始檔直接參考到的原始檔案，這表示說如果你有一個過時的類別檔案，它並未被最新的類別檔案參考到，它就不會被注意到且重新編譯了。你可以用 -depend 選項來強迫 javac 檢查全部的物件，但是注意，這會大量增加編譯的時間；此外，如果這些類別完全沒有被參考到，這個技巧仍然無法幫助你保有最新的類別函式庫或其他類別的集合。這時你應該考慮用 make 公用工具了。

最後，有一點你需要注意的重點是，就算被參考類別只有編譯過的版本，javac 還是可以編譯一個程式，你不需要所有物件的原始檔案。Java 類別檔案包含所有資料型態和原始檔案中的方法特徵資訊，所以編譯二位元的類別檔就像編譯原始檔案一樣的型態安全（而且是例外安全的）。

JAR

Java 壓縮檔案（Java archive file，JAR 檔案）就像 Java 的行李箱一樣，它們將你程式中的每一部分壓縮到一個緊密包裹中，以便於發送或安裝。JAR 是標準且簡單的做法，你可以在 JAR 檔案中放進任何你想要的東西：Java 類別檔案、序列化的物件、資料檔案、圖片、聲音等等。我們在第二十章會看到，一個 JAR 檔案可以有一個或數個數位簽章（digital signature），用以驗證資料的完整性和可靠性。簽章可以加到檔案中，或是加到檔案中的某一個項目。

Java 執行時期系統認識 JAR 檔案，而且可以直接從壓縮檔中載入類別檔案，所以你可以將你程式中的類別壓縮在一個 JAR 檔中，並且把它放在你的 CLASSPATH 上。對於 applet，將它的 JAR 檔案列在 HTML 的 <APPLET> 標籤中的 ARCHIVE 屬性，也會有一樣的效果。若在 JAR 檔中包含的其他類型檔案（資料、圖片等），可以使用 getResource() 方法將它們讀出（在第十章會介紹）。因此，你的程式不用知道資料來源是一般檔案或是 JAR 壓縮檔。一個已知的類別或資料檔案不論是 JAR 檔案中的一個項目、類別路徑上的個別檔案、或是在遠端 applet 伺服器上，你永遠都可以用一種標準方式去參考到它，並且只要讓 Java 的類別載入器去解析位置就行了。

檔案壓縮

要存放在 JAR 檔案內的項目可以使用 ZLIB 來壓縮【註】。JAR 檔案和視窗使用者所熟悉的 ZIP 壓縮檔是完全相容的，你甚至可以使用像 pkzip 之類的工具來建立或維護簡單的 JAR 檔，但是 Java 的壓縮工具 jar 所提供的功能更加強大。

壓縮使得經由網路下載類別的速度更快。據估計，SDK 發行軟體的一個典型類別檔案在壓縮後小了 40%。文字檔案，如：任意的 HTML、內含英文字的 ASCII 檔，通常在壓縮後可以達到原本大小的 25%。（但是，圖片檔在壓縮後並不會變小，因為一般的圖片格式都已經是壓縮檔了。）

壓縮不止是有利於 JAR 檔在網路上傳輸。對一個有許多元件的應用程式而言，連接設定和發出請求的時間遠多於傳輸所有元件所需的時間，所以 JAR 對於經由網路傳輸的 applet 而言特別重要。典型的網頁瀏覽器必須對每一個類別或檔案發出個別的 HTTP 請求，舉例來說，一個含有 100 個類別的 applet 至少要對網頁伺服器發出 100 個請求才可以收集到所有元件。將所有的類別放入到一個 JAR 檔，使得一次傳輸就完成全部類別的下載。減少發出 HTTP 請求的成本可以省下很多的時間，因為個別的類別檔案通常很小，而且一個複雜的 applet 通常都需要很多的類別。

註　參考 http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1950.html 和 RFC 1950。

jar

SDK 提供的 jar 是建立和讀取 JAR 檔案的簡便工具，它的使用者介面不是很容易親近，它模仿 Unix 的 tar 命令（tar 是 tape archive 的簡稱）。如果你熟悉 tar，你就會認得下面的命令：

`jar -cvf` *`jarFile path`* `[` *`path`* `] [ ... ]`

建立包含 *path* 的 *jarFile*

`jar -tvf` *`jarFile`* `[` *`path`* `] [ ... ]`

列出 *jarFile* 的內容，可選擇只列出位在 *path* 中的

`jar -xvf` *`jarFile`* `[` *`path`* `] [ ... ]`

解壓 *jarFile* 的內容，可選擇只有在 *path* 中的

這些命令中的字母 c、t、x 是告訴 jar 要建立壓縮檔、列出壓縮檔內容、或解開壓縮檔中的檔案。f 代表下一個參數要處理的 JAR 檔案名稱。v 則告訴 jar 在顯示檔案壓縮資訊時候要使用詳細資料模式，在詳細資料模式中，你可以得到檔案的大小、修改時間、壓縮比率等資訊。

之後的項目（就是在 jar、字母、要處理的檔案 *jarFile* 之後的所有項目）都被視為壓縮檔項目的名稱。如果你是建立一個壓縮檔，你列出的檔案和目錄就會放在它下面。如果你是解壓檔案，則只有你列出的檔案會從壓縮檔中被解開（如果你沒有列出任何檔案，jar 就會解開壓縮檔的所有檔案）。

例如，我們剛剛完成了一個新遊戲叫 "spaceblaster"，所有和這遊戲相關的檔案都放在三個目錄中：Java 類別本身放在 spaceblaster/game 目錄中，遊戲的圖片檔放在 spaceblaster/images 目錄中，遊戲相關資料則是放在 spaceblaster/docs 目錄中。我們將所有東西都用以下這個命令壓縮成一個壓縮檔：

```
% jar -cvf spaceblaster.jar spaceblaster
```

因為我們要求詳細的結果，jar 會顯示它做了什麼事：

```
adding:spaceblaster/ (in=0) (out=0) (stored 0%)
adding:spaceblaster/game/ (in=0) (out=0) (stored 0%)
adding:spaceblaster/game/Game.class (in=8035) (out=3936) (deflated 51%)
```

```
adding:spaceblaster/game/Planetoid.class (in=6254) (out=3288) (deflated 47%)
adding:spaceblaster/game/SpaceShip.class (in=2295) (out=1280) (deflated 44%)
adding:spaceblaster/images/ (in=0) (out=0) (stored 0%)
adding:spaceblaster/images/spaceship.gif (in=6174) (out=5936) (deflated 3%)
adding:spaceblaster/images/planetoid.gif (in=23444) (out=23454) (deflated 0%)
adding:spaceblaster/docs/ (in=0) (out=0) (stored 0%)
adding:spaceblaster/docs/help1.html (in=3592) (out=1545) (deflated 56%)
adding:spaceblaster/docs/help2.html (in=3148) (out=1535) (deflated 51%)
```

jar 產生 spaceblaster.jar 檔案，並新增 spaceblaster 目錄，再將 spaceblaster 內的目錄和檔案加入到壓縮檔中。在詳細資料模式，jar 會記錄壓縮所節省的空間。

我們可以用這個命令解開壓縮檔：

```
% jar -xvf spaceblaster.jar
```

同樣的，我們可以解開個別檔案或是目錄：

```
% jar -xvf spaceblaster.jar filename
```

但你通常不需要自己解開 JAR 檔去使用它的內容，Java 工具會知道如何從壓縮檔中讀取資料。我們可以用這個命令去列出 JAR 的內容：

```
% jar -tvf spaceblaster.jar
```

底下是輸出結果，它列出所有的檔案、大小和產生時間：

```
     0 Thu May 15 12:18:54 PDT 1997 META-INF/
  1074 Thu May 15 12:18:54 PDT 1997 META-INF/MANIFEST.MF
     0 Thu May 15 12:09:24 PDT 1997 spaceblaster/
     0 Thu May 15 11:59:32 PDT 1997 spaceblaster/game/
  8035 Thu May 15 12:14:08 PDT 1997 spaceblaster/game/Game.class
  6254 Thu May 15 12:15:18 PDT 1997 spaceblaster/game/Planetoid.class
  2295 Thu May 15 12:15:26 PDT 1997 spaceblaster/game/SpaceShip.class
     0 Thu May 15 12:17:00 PDT 1997 spaceblaster/images/
  6174 Thu May 15 12:16:54 PDT 1997 spaceblaster/images/spaceship.gif
 23444 Thu May 15 12:16:58 PDT 1997 spaceblaster/images/planetoid.gif
     0 Thu May 15 12:10:02 PDT 1997 spaceblaster/docs/
  3592 Thu May 15 12:10:16 PDT 1997 spaceblaster/docs/help1.html
  3148 Thu May 15 12:10:02 PDT 1997 spaceblaster/docs/help2.html
```

JAR 清單

注意 jar 加了一個 META-INF 目錄到我們的壓縮檔，目錄中包含一個檔案：MANIFEST.MF。META-INF 目錄中存放描述這個 JAR 檔案內容的檔案。MANIFEST.MF 檔案是 jar 自動產生的，它是一份清單（manifest），清單中包含這個 JAR 檔中所有的壓縮檔案，以及每個檔案的和總檢查碼（checksum）。

清單是一個文字檔案，包含數行關鍵字:值（keyword:value）這樣形式的資料。清單的格式在 SDK 1.1 和 SDK 1.2 之中有所不同，在 SDK 1.2 和其後的版本，清單變的非常簡單，沒有包含壓縮檔內任何項目的資訊：

```
Manifest-Version: 1.0
Created-By: 1.2.1 (Sun Microsystems Inc.)
```

基本上，這個檔案只是描述它的版本號碼。在 SDK 1.1 中，清單包含這個壓縮檔中所有項目的描述。在我們的例子中，我們的清單檔案開頭看起來像是這個樣子（這只有在 SDK 1.1 中）：

```
Manifest-Version: 1.0

Name: spaceblaster/game/Game.class
Digest-Algorithms: SHA MD5
SHA-Digest: D5Vi4UV+O+XprdFYaUt0bCv2GDo=
MD5-Digest: 9/W62mC4th6G/x8tTnP2Ng==

Name: spaceblaster/game/Planetoid.class
Digest-Algorithms: SHA MD5
SHA-Digest: SuSUd6pYAASO5JiIGlBrWYzLGVk=
MD5-Digest: KN/4cLDxAxDk/INKHi2emA==

...
```

第一行和 SDK 1.1 一樣是版本號碼，接著數行一組是每個項目的描述。第一行是項目的名稱，此例中，分別是 Game.class 和 Planetoid.class 檔案。每一組的其餘幾行是描述這個項目的各種屬性，此例中，Digest-Algorithms 這一行指出這個清單提供 SHA 和 MD5【註】兩種訊息編碼（message digest，類似和總檢查碼）。接著是這個項目使用這兩種演算法的實際訊息編碼。

註　SHA 是 Secure Hashing Algorithm，MD5 是 Message Digest 5，你只需要知道這些就可以了，演算法的解釋超過本書的探討範圍。

我們下一節將會討論到，在 JAR 檔中的 META-INF 目錄和清單檔案也可以有項目的數位簽章訊息。訊息編碼資訊只有對已簽章的 JAR 檔案才真正需要，當你在 SDK 1.2 和之後的版本建立壓縮檔時，它就被省略了。

當你建立一個壓縮檔時，你可以指定補充的清單檔案到清單描述中，藉此來加入你自己的資訊，這是在壓縮檔中存放檔案其他種簡單屬性資訊的好地方。

例如，我們使用以下的「關鍵字:值」來建立一個檔案：

```
Name: spaceblaster/images/planetoid.gif
RevisionNumber: 42.7
Artist-Temperment: moody
```

若想把這個資訊加到壓縮檔目錄的清單中，可以把它存在檔案中，將檔案命名為 myManifest.mf，然後用底下的 jar 命令加入：

```
% jar -cvmf myManifest.mf spaceblaster.jar spaceblaster
```

我們在這個命令放了另一個選項 m，它指示 jar 應該從命令列中給定的檔案去讀取額外的清單資訊。jar 如何知道該讀那個檔案？因為 m 在 f 之前，所以它預期會在它產生的 JAR 檔名之前可以找到清單資訊。如果你覺得這種做法很笨拙，那麼你就對了，萬一名字的順序錯誤，那麼 jar 也會跟著做錯，請務必小心！

除了你自己使用的訊息，還可以放些有用的特殊值在清單檔案中（在 SDK 1.2），其中之一就是 Main-Class，它許允你指定含有 main() 方法的類別：

```
Main-Class: Game
```

如果你合併使用這二樣技術（Main-Class 和 m 選項），你可以從命令列執行 JAR：

```
% java -jar spaceblaster.jar
```

直譯器會在清單中尋找 Main-Class 值，然後載入指定名稱的類別作為這個程式的初始類別。

我們在 JAR 檔中加入的聰明資訊和辛苦的修訂版有什麼用呢？非常不幸的，除了解壓縮檔案和讀取清單外，一點用都沒有！但是如果你想要寫你自己的 JAR 工具或是某種資源載入器，你可以加入一些程式碼去察看清單、檢查你個人的關鍵字，然後再做出相對的回應，例如，當關鍵字「心情」的值是「鬱悶」時，就顯示灰暗的色彩。

另一個重要的關鍵字是 Java-Bean。如果這個項目的是 Java Bean，則這個關鍵字的值應該為 true；這個資訊主要是由 BeanBox 和其他和 Bean 有關的公用工具所使用（參閱第十九章）。