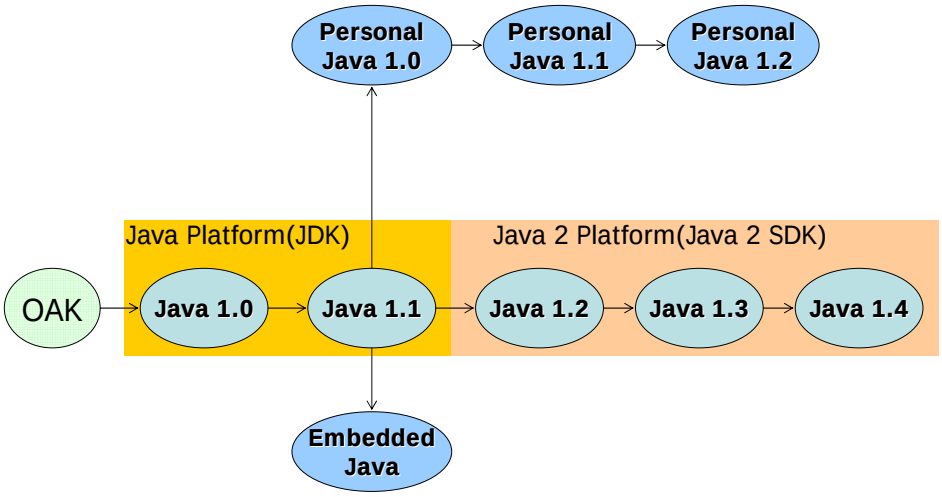
# 第一章
# 深入 Java 2 SDK

你越討厭的事情，就越容易遇上

## 前言

Sun Microsystems 所發表的 Java 開發工具 –Java 2 SDK，永遠都是 Java 初學者最早接觸到的開發工具。一般人習慣稱這套工具叫作 JDK(Java Development Kit)。

**圖:Java 版本及開發工具的演進**

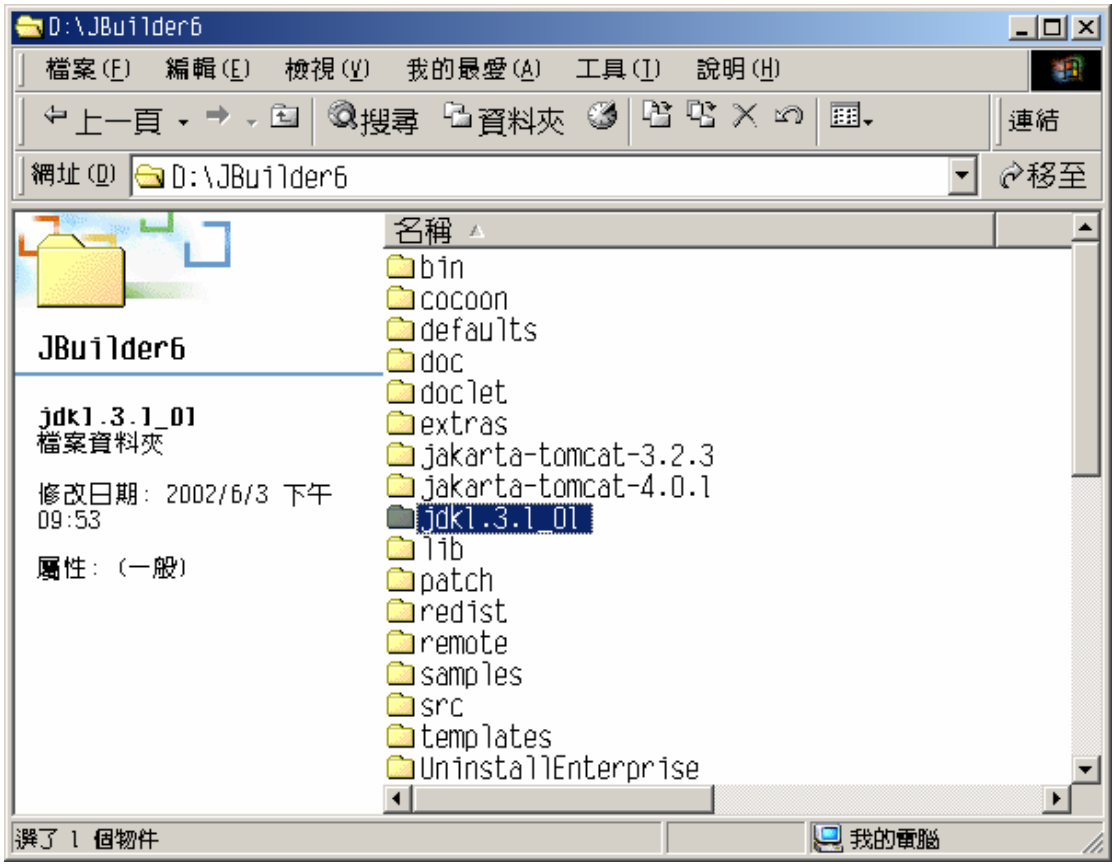



JDK 是在 Java 2 Platform 之前的 Java Platform 所使用的開發工具名稱(在本文中有時候會寫 JDK，有時候會寫 Java 2 SDK，但指的將是同一種東西)，我記得曾經有人戲稱它是 Java Developer Killer 的縮寫，除了挖苦 Sun 所製作的開發工具沒有微軟設計的開發工具要來的方便之外，其實還說明了另外一件事，就是 Java 從 1995 年發表後到現在，即使每一版的 JDK 都會附上為數龐大的官方說明文件，從來沒有任何一份文件或一本書籍詳細說明這套官方開發工具的特性。

沒有文件或書籍來描述 JDK 的特性並不代表這些特性不重要。畢竟，任何最新的標準類別函式庫，或是最新版本的虛擬機器，一定都會伴隨著最新版的 JDK 所釋出。就算您想跳過 JDK，直接使用如 Borland JBuilder 或 Forte for Java 這類的高級開發工具，JDK 仍然如影隨形。以 Borland JBuilder 來說，當您將 Borland JBuilder 安裝完成之後，在 JBuilder 的所在目錄下也會內含一套 Java

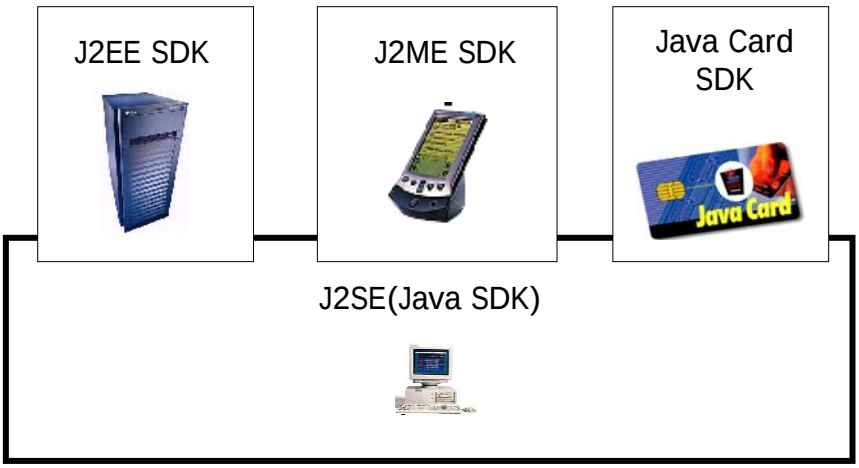
2 SDK，如下圖所示:

**圖:Borland JBuilder 內附的 Java 2 SDK**




因此，我們可以得知，不管您如何地討厭 JDK，只要想開發 Java 相關的應用程式，您就無法逃離 JDK 的掌握。事實上，不管您開發的是 J2SE、J2EE、J2ME、甚至 Java Card 的應用程式，除了需要各種版本對應的開發套件之外，一定需要 JDK 的輔助。

**圖:各種版本的 Java 應用程式，都需要 Java 2 SDK 的輔助**




那麼，JDK 到底是什麼東西? 從技術的觀點來說，因為高階開發工具都是架設在 JDK 上頭，因此高階開發工具的行為或是引發的錯誤訊息都是根源自 JDK。為了更正確地掌控高階的 Java 開發工具，所以我們必須了解 JDK 的特性和組成。 從求知的觀點來看，Java 程式設計師每天輸入無數的 java xxx.java 與 java xxx，到底 Java 程式是如何運作的? 在我們看不到的底層，到底發生了什麼事情? 如果我們可以清楚地得知所有的來龍去脈，將會讓我們更了解這套開發工具。

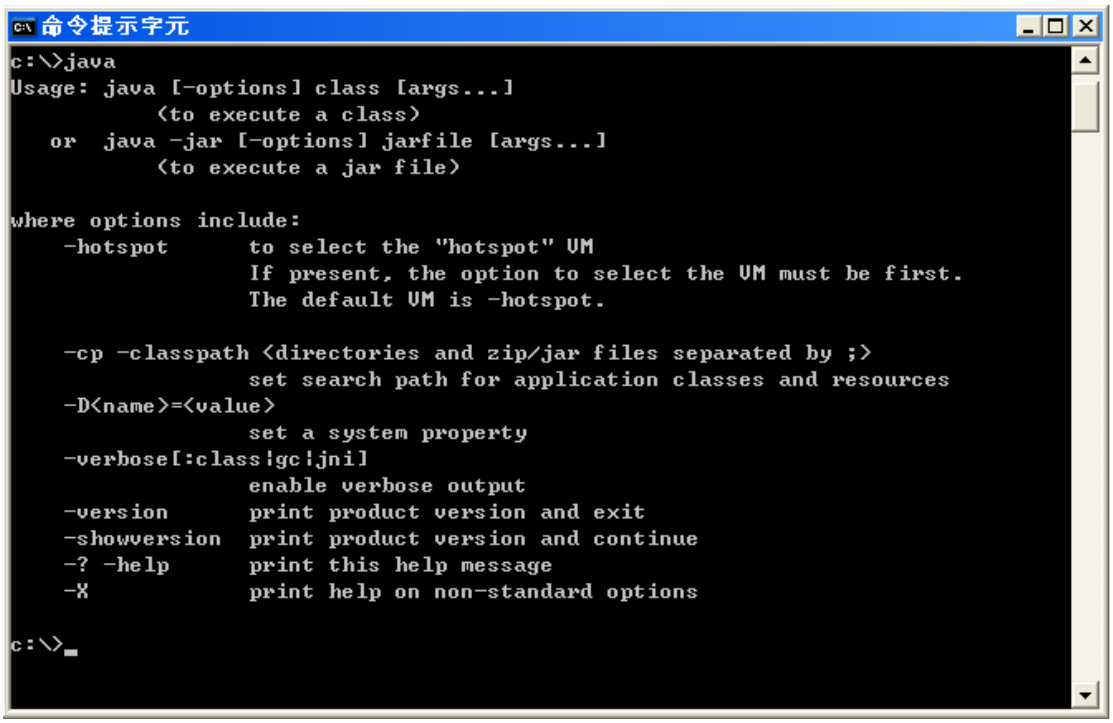
上述兩個觀點，都是本章所希望告訴您的。讓我們開始深入了解 JDK 吧!

## 執行 java.exe 時所發生的怪事

當您在使用 JDK 時，您是否曾經發現執行 java.exe 的時候，會有底下一些奇怪的現象:

**如果您安裝的是 Java 2 SDK 1.3.x :**

當您安裝完 Java 2 SDK 1.3.x 之後，如果從未修改您電腦裡頭的任何設定，就直接進到命令提示字元下，執行 java.exe，就會出現底下畫面:



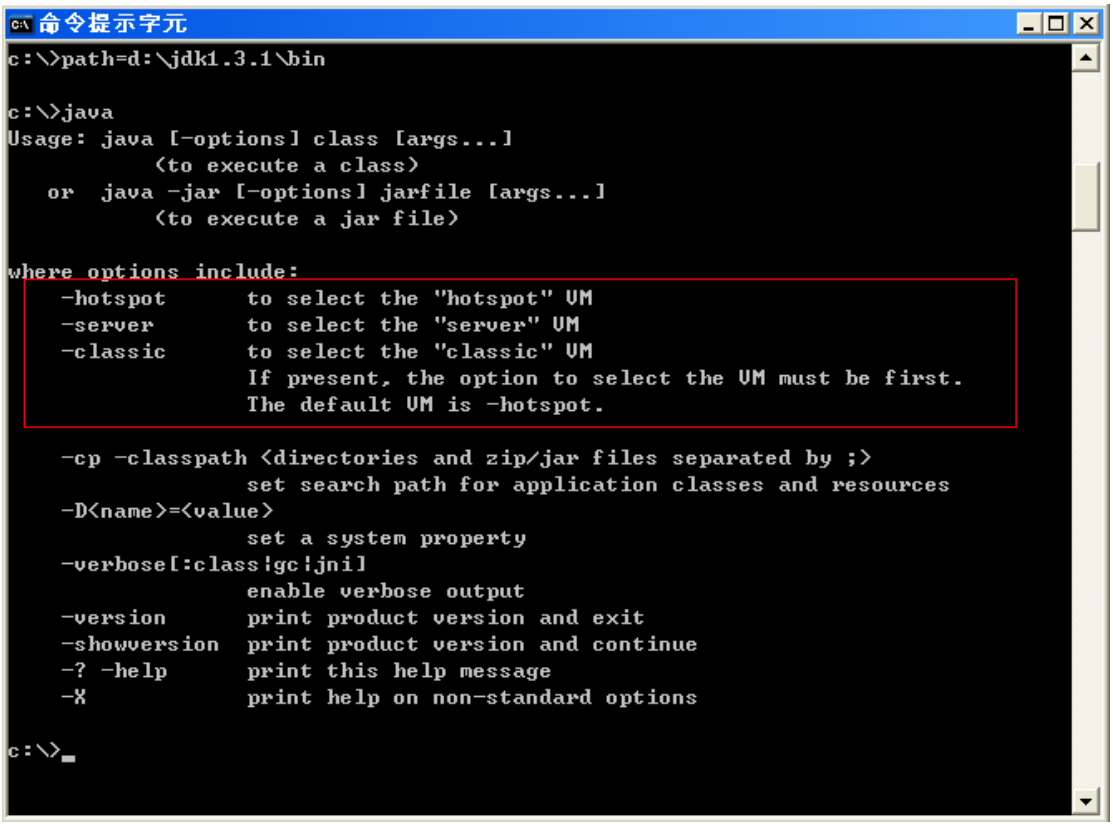

螢幕上會告訴您，有一個 **–hotspot** 選項可供您使用。

但是，此時如果我們輸入指令:

**path=c:\jdk1.3.1\bin**

(註:假設筆者的 Java 2 SDK 1.3.x 安裝於 c:\jdk1.3.1 底下)

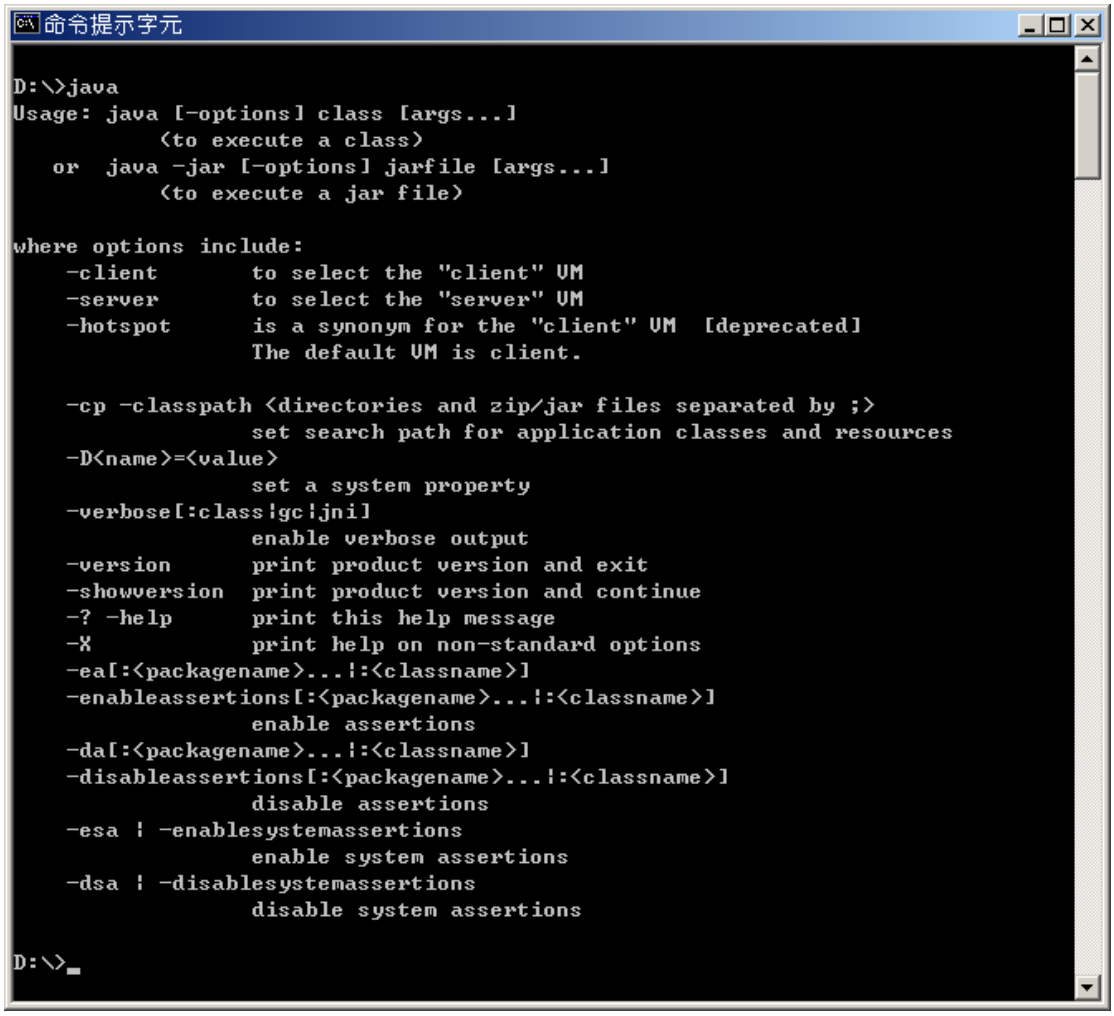
然後重新執行 java.exe，則螢幕上的輸出如下:



這一次，螢幕上的輸出告訴我們有 –hotspot、-server、以及-classic 可供選擇。這真是奇怪的事情，原本根本沒有出現在螢幕上的–hotspot、-server、以及-classic 選項，在我們使用 path 這個系統命令來改變執行檔的搜尋路徑之後，竟然出現了!

**如果您安裝的是 Java 2 SDK JDK 1.4.x :**

當您安裝完 Java 2 SDK 1.4.x 之後，如果從未修改您電腦裡頭的任何設定，就直接進到命令提示字元下，執行 java.exe，就會出現底下畫面:

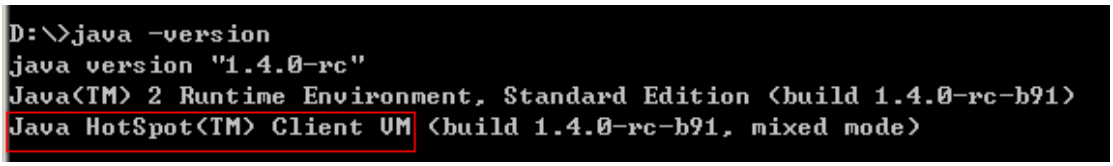



畫面上告訴您，java.exe 有幾個選項，包括 –client、-server、-hotspot(與-client 意義相同，但是已不建議使用)。這幾個選項可以用來調整執行 Java 程式時所使用的 Java 虛擬機器。

所以此時如果您輸入

**java –version**

的時候，螢幕輸出如下:



而如果輸入:

**java –server -version**

並執行之，螢幕上會出現錯誤訊息:

**Error: no `server' JVM at `C:\Program Files\Java\j2re1.4.0-rc\bin\server\jvm.dll`.**

意思是說，它在特定位置找不到名為 server 的虛擬機器。也就是雖然螢幕上有列出此選項，可是卻無法使用。

但是，此時如果我們輸入指令:

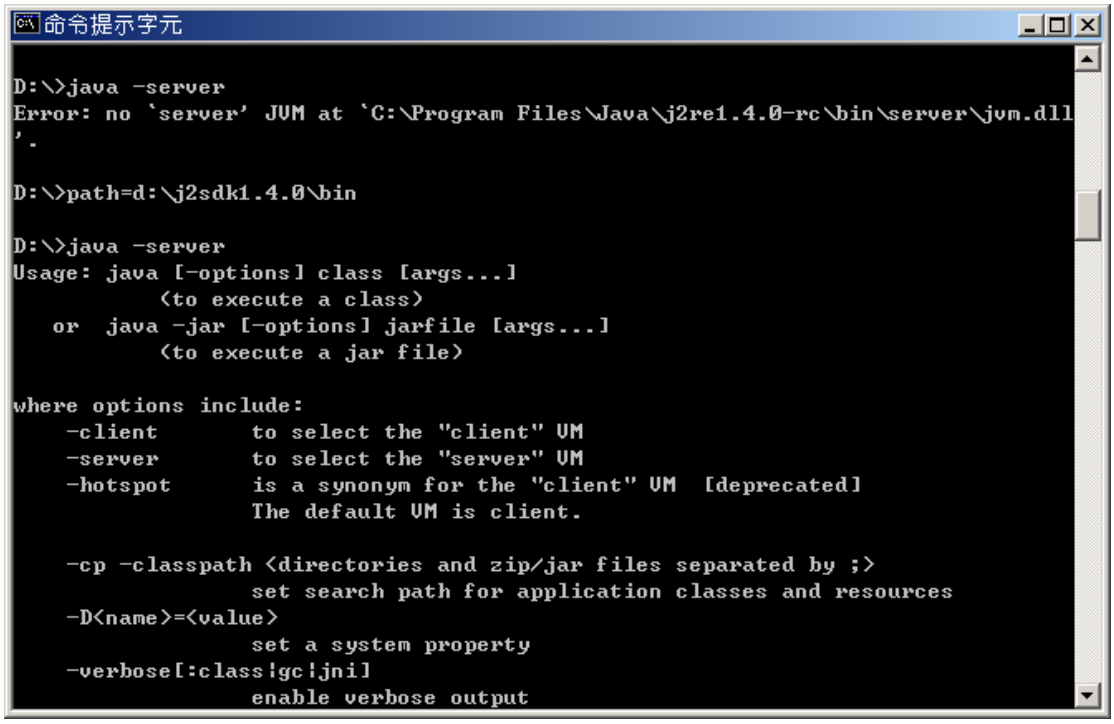
**path=d:\j2sdk1.4.0\bin**

(註:假設筆者的 JDK 1.4.x 安裝於 d:\j2sdk1.4.0 底下)

然後重新執行:

**java –server**

將不再出現錯誤訊息:




如果此時您輸入

**java –server -version**

則螢幕的輸出如下:

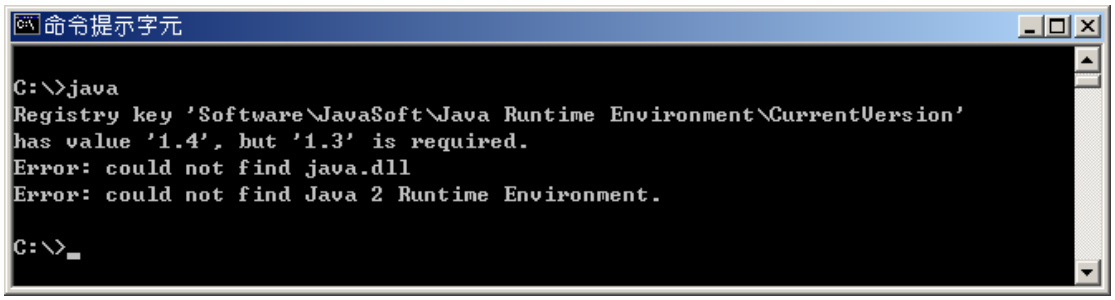

真奇怪，原本無法運作的-server 選項，在我們使用 path 這個系統命令來改變執行檔的搜尋路徑之後，竟然可以運作了!

這真是令人匪夷所思的事情。

如此怪異的現象，在 JDK 1.3.x 上最容易看出來(JDK 1.4.x 上其實也是一樣奇怪，因為**無法使用的選項**以及**沒有出現在我們眼中的選項**，基本上是一樣的。只不過如果沒有特別去嘗試，就很難看得出來，搞不好您還會認為眼不見為淨，出現一些無法使用的選項反倒混淆使用者):

上述這麼奇特的現象，該做如何解釋? 我們有辦法預測其行為嗎? 再者，在您過去使用 Java 2 SDK 來開發 Java 應用程式的生涯之中，是否也曾發生過底下的怪事:

很可能您才剛剛灌好某個版本的 Java 2 SDK 或剛剛移除某個版本的 Java 2 SDK，但是之後卻在執行 java.exe 時，螢幕上竟然告訴您:




在您經驗中，螢幕上的 1.4 和 1.3 可能會換成其他版本號碼，總之就是版本不合所引發的問題。

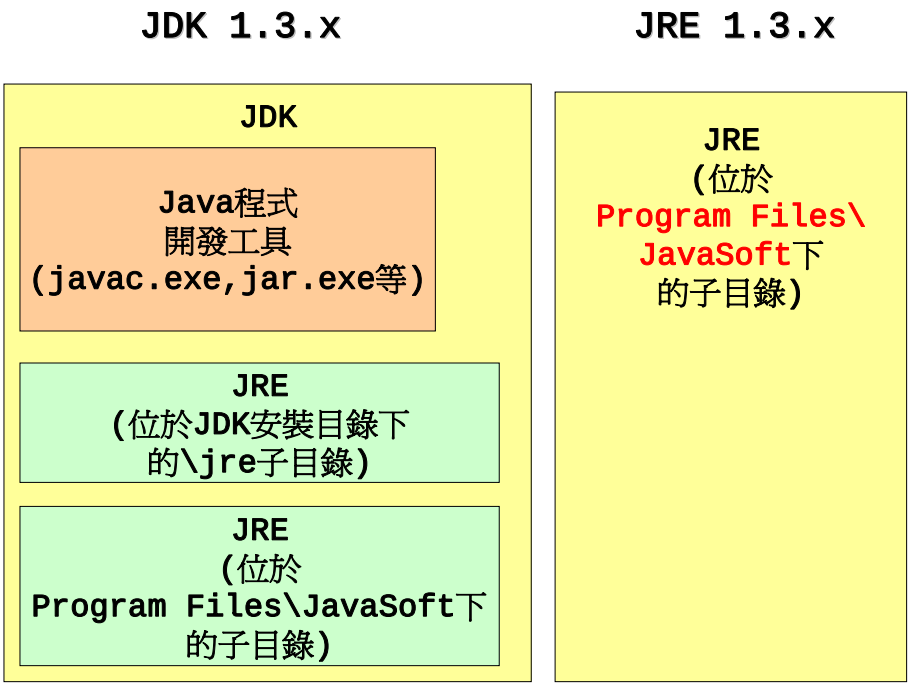
因此接下來，我們將開始探討這種"靈異現象"的成因，並追蹤其來龍去脈。最後，當您下次再遇到這種情況時，您將可以很容易地找出問題的解決方法。

## JDK、JRE、JVM 之間的關係

要探討上述的奇怪現象，必須先弄清楚 JDK(Java 開發套件)、JRE(Java 執行環境，Java Runtime Environment)、還有 JVM(Java 虛擬機器，Java Virtual Machine)三者之間的關係。

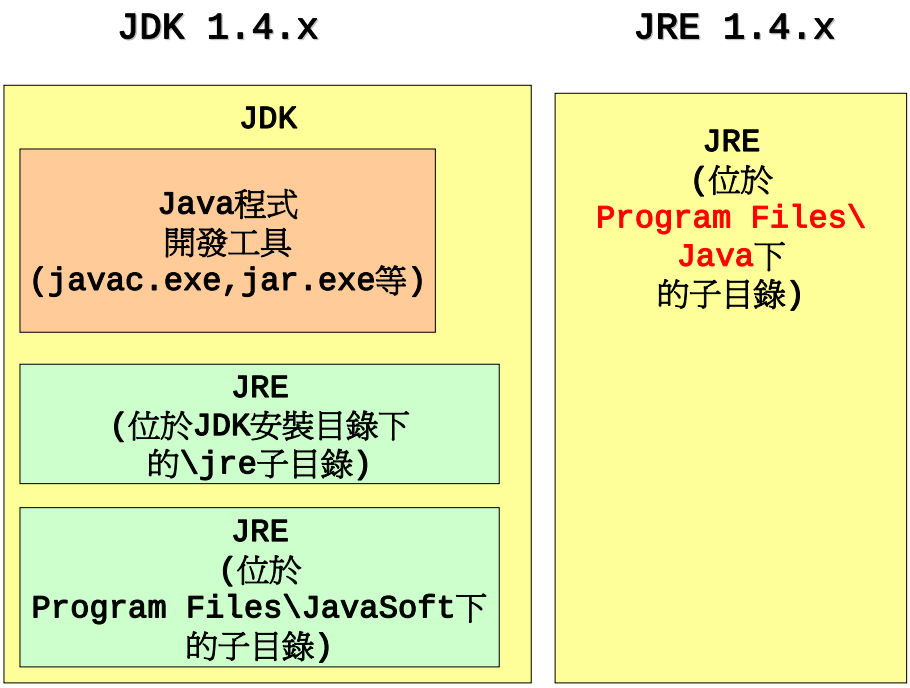
當您想要從 http://www.javasoft.com 下載 JDK 來開發程式的時候，或許您曾經疑惑過，官方網站上會出現 JDK 與 JRE 兩種套件，然而一般初學者不清楚 JDK 和 JRE 有什麼不同，所以產生疑惑。這兩者一定有所不同，否則 Sun Microsystems 不可能將兩者區分開來。關於 JDK 與 JRE 兩者間的關聯，我們可以用底下圖片來描述:

**圖:JDK 1.3.x 與 JRE 1.3.x 的關係**




JDK 1.4.x 和 JDK 1.3.x 之間有些許的不同，如下圖所示:

**圖:JDK 1.4.x 與 JRE 1.4.x 的關係**

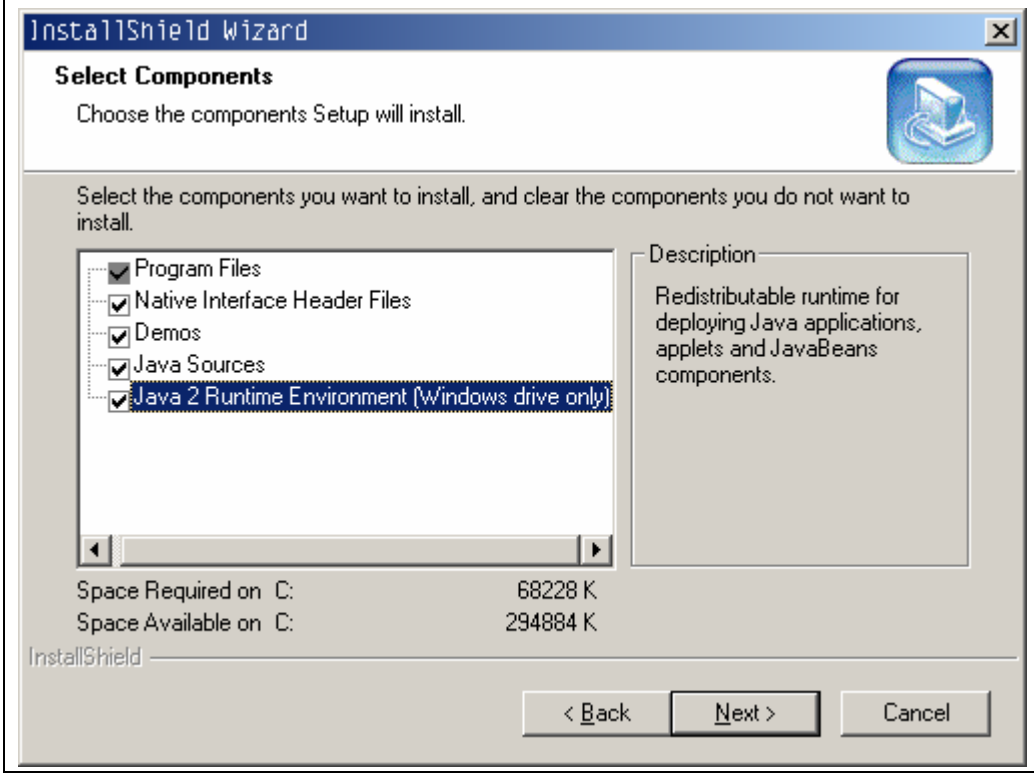



從上圖可以得知，如果您裝了 JDK，那麼您的電腦底下一定會有兩套 JRE、一套位於 **<jdk 安裝目錄>\jre** 底下(JDK 1.3.x 與 1.4.x 皆是如此)，另外一套位於 **C:\Program File\JavaSoft** 底下(JDK 1.4.x 則是放在 **C:\Program File\Java** 底下)。

| 注意! |
|---|

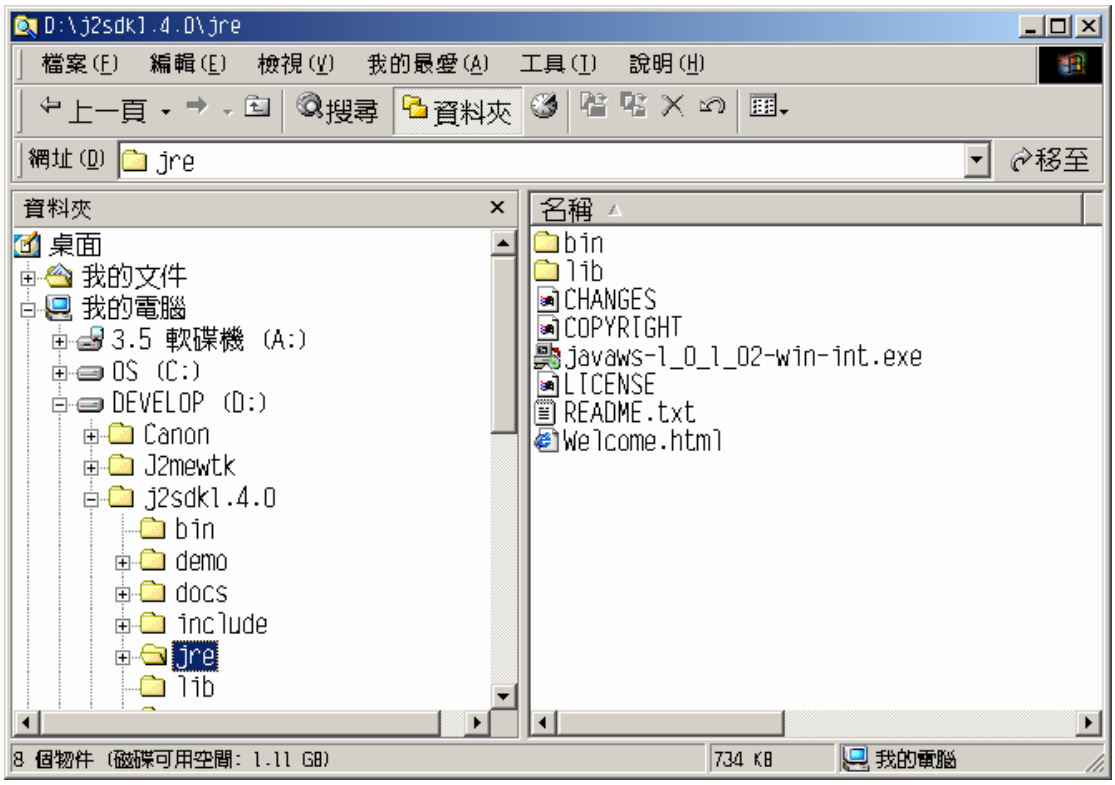
如果您安裝的是 JDK 1.3.x，那麼您沒有選擇的餘地，安裝程式一定會同時安裝兩套 JRE 在您的電腦上。

可是如果是安裝 JDK 1.4.x 的話，您可以在安裝 JDK 時選擇是否安裝位於 **C:\Program File\Java** 目錄下的 JRE，但是**<jdk 安裝目錄>\jre** 底下的這套 JRE 一樣必須安裝，沒有選擇的餘地。下圖是 JDK 1.4.x 的安裝畫面，您可以看到 JRE 是否安裝可以由您自己決定，不過它指的 JRE 是位於 **C:\Program File\Java** 目錄下的 JRE，千萬別跟**<jdk 安裝目錄>\jre** 底下那套 JRE 混淆了。



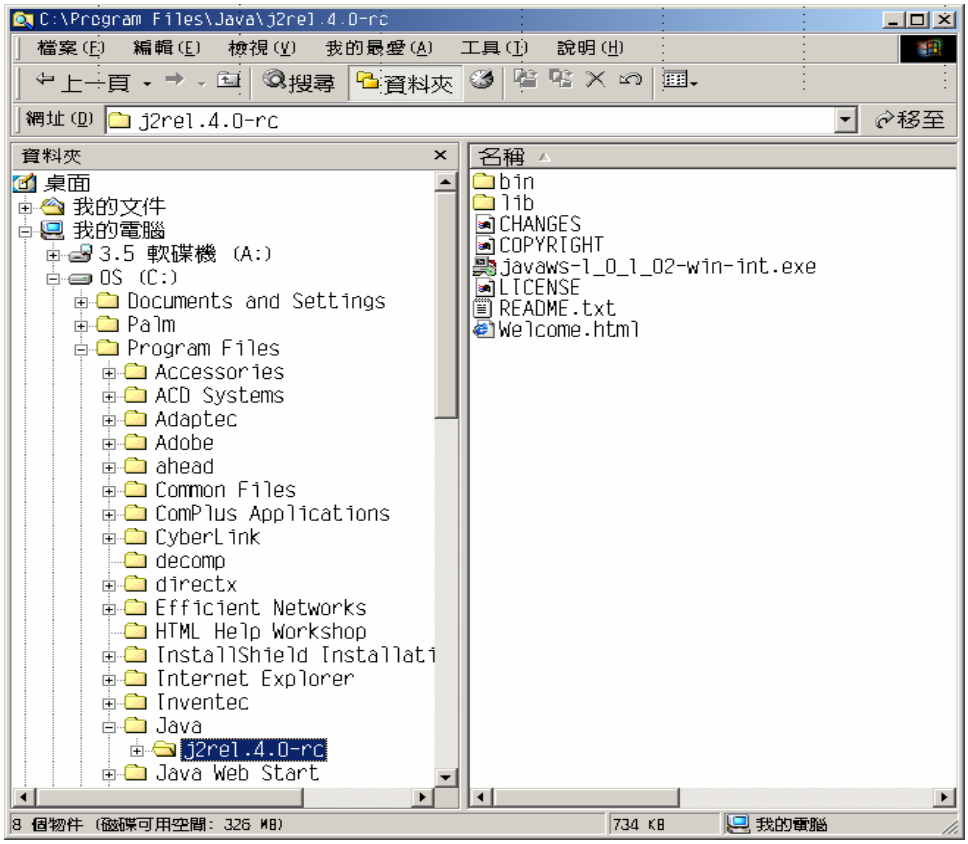

您的電腦上安裝了 JDK 1.4.x 之後，檔案系統中的檔案分佈如下圖:

**圖:安裝 JDK 1.4.x 之後，系統中的檔案分佈情形(1)**




與

**圖:安裝 JDK 1.4.x 之後，系統中的檔案分佈情形(2)**




不曉得大家有沒有發現，兩個目錄下所出現的檔案和子目錄根本一模一樣(其實

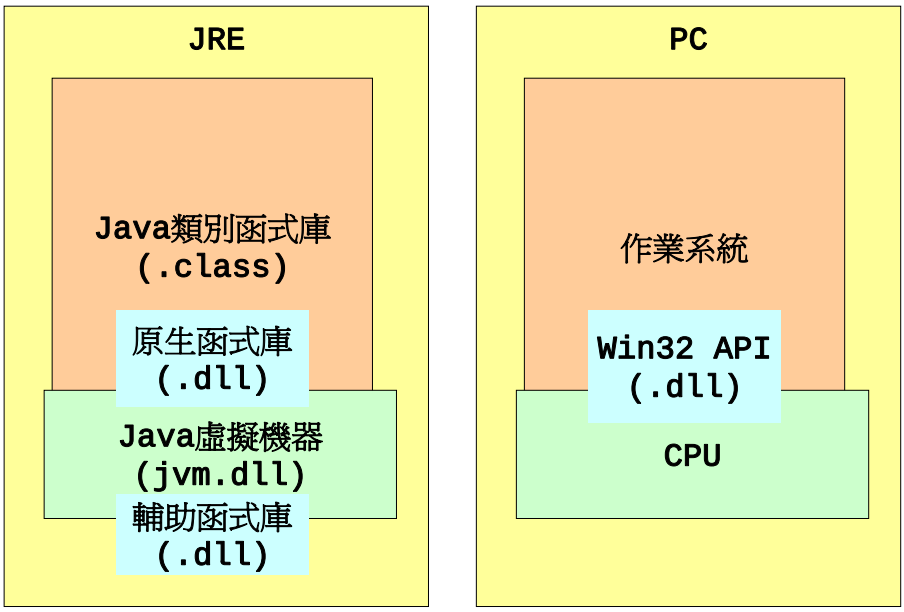
有很小很小的差別，但是對我們的討論沒有任何影響，因此本文中當作兩者沒有任何差別)。

如果您只從網路下載了 JRE 而非 JDK，那麼就只會在 **C:\Program File\JavaSoft** 底下(JRE 1.4.x 則是在 **C:\Program File\Java** 底下)安裝唯一的一套 JRE。

接下來您一定會問，那麼 JRE 是做什麼用的? 筆者用下圖描述 JRE 的功能:

**圖:JRE 與 PC 的類比**

**JRE與PC**




從上圖您可以知道，JRE 的地位就像一台 PC 一樣，我們撰寫好的 Win32 應用程式需要作業系統幫我們執行，同樣地，我們所撰寫的 Java 程式也必須要 JRE 才能幫我們執行。所以。當您裝完 JDK 之後，如果分別在硬碟的不同地方安裝了兩套 JRE，那麼您就可以想像您的電腦有兩台虛擬的 Java PC，都具有執行 Java 應用程式的功能。所以 Java 虛擬機器只是 JRE 裡頭的其中一個成員而已，以更技術的角度來說，Java 虛擬機器只是 JRE 裡頭的一個動態聯結函式庫罷了。

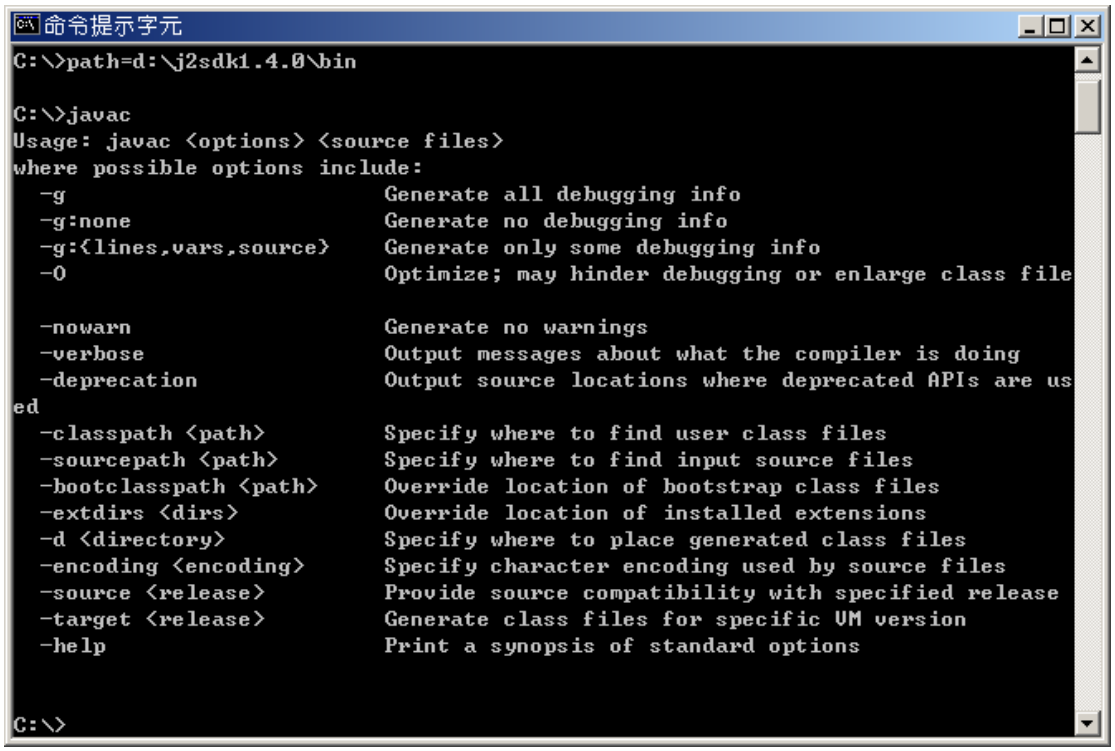
所以我們可以說，只要您的使用者電腦之中安裝了 JRE，就可以正確地執行 Java 應用程式(Windows XP 宣稱不支援 Java，就是因為微軟不再於 Windows XP 中隨機附上自己版本的 JRE，不過，就算微軟不附自己的 JRE，Sun 還是提供可以支援 Windows XP 的 JRE，JDK 1.4.x/JRE 1.4.x 就完全支援 Windows XP)。

接下來您一定會問，那麼 Sun 為何要讓 JDK 的安裝程式同時安裝兩套幾乎完全相同的 JRE 呢? 花費額外的硬碟空間在不同的目錄下安裝兩個一模一樣的東西，頗有浪費之嫌，不是嗎? **其實真正的原因是因為 - JDK 裡面也附上了很多用 Java 所撰寫開發工具**(例如 javac.exe、jar.exe 等)，而且他們都放置在

**<jdk 安裝目錄>\lib\tools.jar** 這個檔案之中。

什麼? Java 的編譯器 javac.exe 也是用 Java 撰寫的，真的嗎? javac.exe 明明就是一個執行檔，用 Java 撰寫的程式應該是.class 檔才對。為了證明這件事情，底下我們做個小實驗，證明我所言不假:

首先，請先在命令列模式底下執行底下指令 javac.exe :




然後，請到 **<jdk 安裝目錄>\lib** 底下把 tools.jar 改名成 tools1.jar，然後重新執行 javac.exe，您將看到螢幕輸出如下:


這個意思是說，您輸入 javac.exe 和輸入

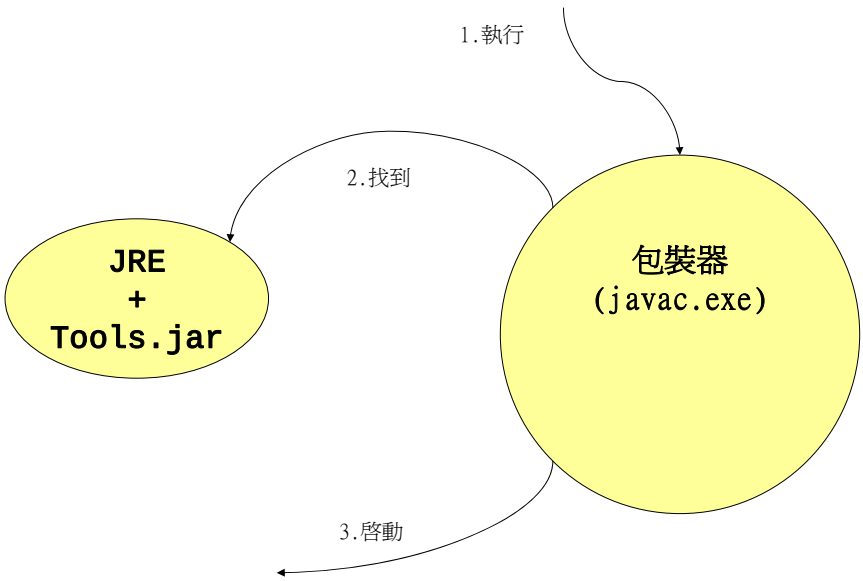
**java -classpath d:\j2sdk1.4.0\lib\tools.jar com.sun.tools.javac.Main**

會得到相同的結果。請把 tools1.jar 改回成 tools.jar，然後重新執行指令:

```
命令提示字元

C:\>java -classpath d:\j2sdk1.4.0\lib\tools.jar com.sun.tools.javac.Main
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -O                         Optimize; may hinder debugging or enlarge class file

  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are us
ed
  -classpath <path>          Specify where to find user class files
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -d <directory>             Specify where to place generated class files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -help                      Print a synopsis of standard options


C:\>_
```

您會看到執行結果和輸入javac.exe一模一樣。從這裡我們可以證明javac.exe只是一個包裝器(wrapper)，而製作目的是為了讓開發者免於輸入太長的指令。包裝器的概念如下圖所示:

**圖:包裝器的概念**



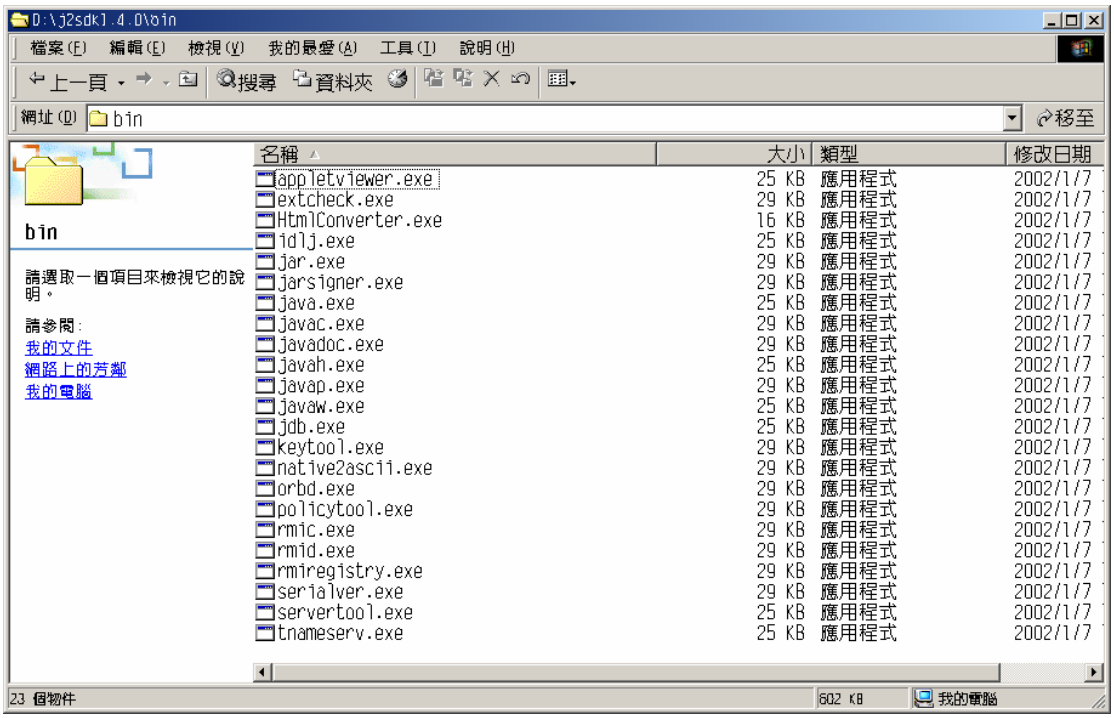

**java -classpath d:\j2sdk1.4.0\lib\tools.jar com.sun.tools.javac.Main**

其實包裝器的概念您也可以在很多地方看到，比方說當您用 J2ME 開發 Palm 應用程式的時候，工具都會幫我們打包 JAR 檔，然後用一個 PRC 檔的外殼罩住，讓 Java 程式看起來像是一個原生(native)的應用程式。如果您開發.NET 應用程式，那麼.NET 開發工具所造出的執行檔也是一個包裝器的概念。不管如何，包

裝器的設計是為了讓使用者更方便地使用您的應用程式。

除了 javac.exe 之外，JDK 很多內附的開發工具也都是採用包裝器的概念。請試著開啟 **<jdk 安裝目錄>\bin** 目錄，您將發現底下的執行檔大小都很小，不大於 29 KB:

**圖:bin 目錄下的執行檔**




從這裡我們可以推得一個結論，就是: **JDK 裡面的工具幾乎是用 Java 所撰寫的，所以 JDK 本身就是 Java 應用程式，因此要使用 JDK 附的工具來開發 Java 程式，也必須要自行附一套 JRE 才行，這就是<jdk 安裝目錄>\jre 底下需要一套 JRE 的原因。而位於 Program File\底下的那套 JRE 就是拿來執行我們自己所撰寫的 Java 應用程式**。不過，兩套中任何一套 JRE 都可以拿來執行我們所撰寫的 Java 應用程式，可是 JDK 內附的開發工具在預設使用包裝器(.exe)來啟動的情形下，都會自己去選用**<jdk 安裝目錄>\jre** 底下那套 JRE。

但是問題也就出在這裡，在同一台電腦之中安裝如此多的 JRE(如果您還安裝了 Borland JBuilder 或 TogetherJ 這些完全使用 Java 開發的軟體，那麼您的電腦裡頭就會有更多套的 JRE)。**一台電腦之中許多套 JRE，再加上作業系統本身 PATH 環境變數的特性，就會造成最開頭所示範的怪異現象:同樣都是執行 java.exe，但是執行結果卻不相同。**

# 您所執行的是哪一個 java.exe ?

既然您的電腦裡頭至少有兩套 JRE，那麼誰來決定用哪一套 JRE 呢? 這個重責大任就落在 java.exe 的身上。

當我們在命令列輸入

**java XXX**

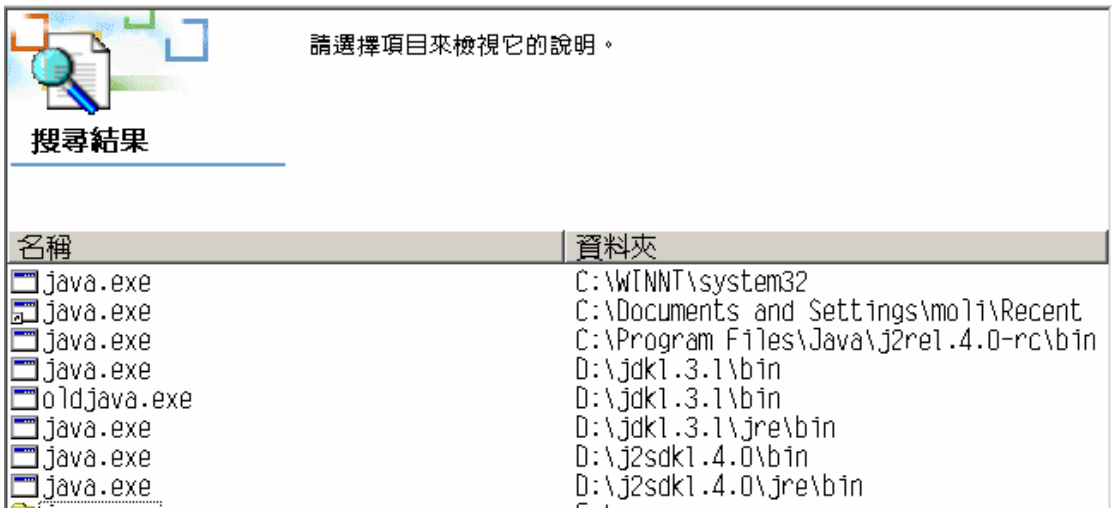
的時候，java.exe 的工作就是找到合適的 JRE 來執行類別檔。java.exe 依照底下邏輯來尋找 JRE:

1. 自己的目錄下有沒有 JRE 目錄。(這個部分這樣說並不是非常精確，原因請詳見 JDK 原始碼，這此不特別說明)
2. 父目錄底下 JRE 子目錄。
3. 查詢 Windows Registry(**HKEY_LOCAL_MACHINE\Software\JavaSoft\Java Runtime Environment**)。

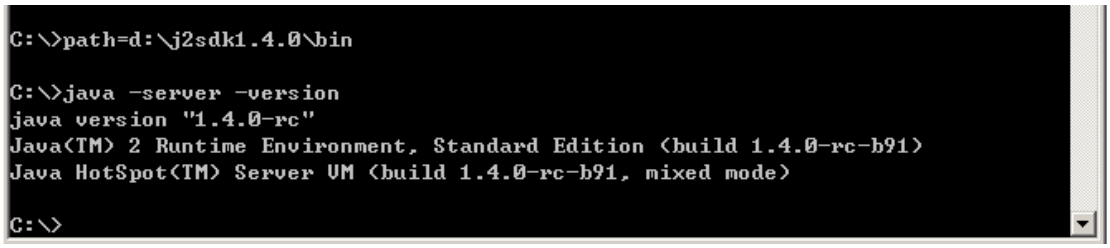
所以，java.exe 的執行結果和您電腦裡面哪一個 java.exe 被執行，然後哪一套 JRE 被拿來執行 Java 應用程式有莫大的關係。

請您先使用 Windows 的搜尋功能，看看您的電腦裡頭到底有多少 java.exe? 筆者的電腦上一經搜尋，找到了好幾個 java.exe，如下圖:

**圖:電腦裡的 java.exe**




所以您可以斷定，如果指令為:




您執行到的 java.exe 一定是 d:\j2sdk1.4.0\bin 底下的 java.exe，所以 java.exe 選到的是父目錄(d:\j2sdk1.4.0)底下的 JRE(d:\j2sdk1.4.0\jre)。

請開啟 d:\j2sdk1.4.0\jre\bin 這個目錄，您會看到 client 和 server 兩個目錄，裡面都會分別看到 jvm.dll，這就是一般我們所謂的 Java 虛擬機器之所在。

**圖:真正的 Java 虛擬機器之所在(hotspot client)**

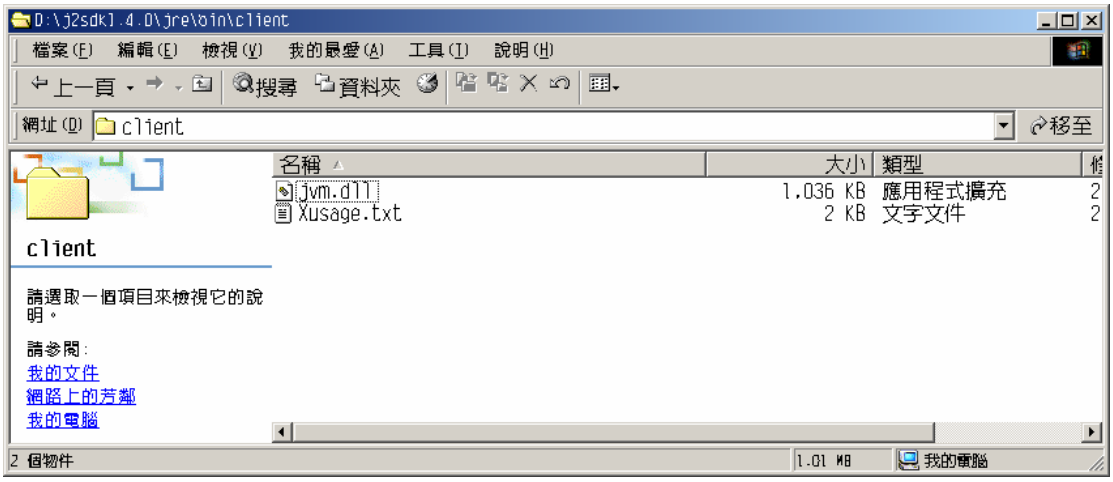
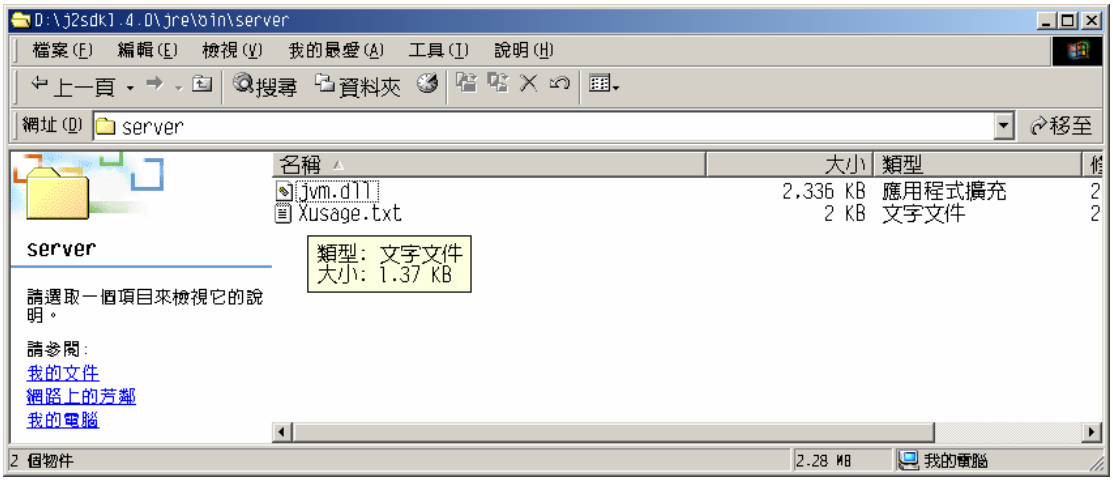


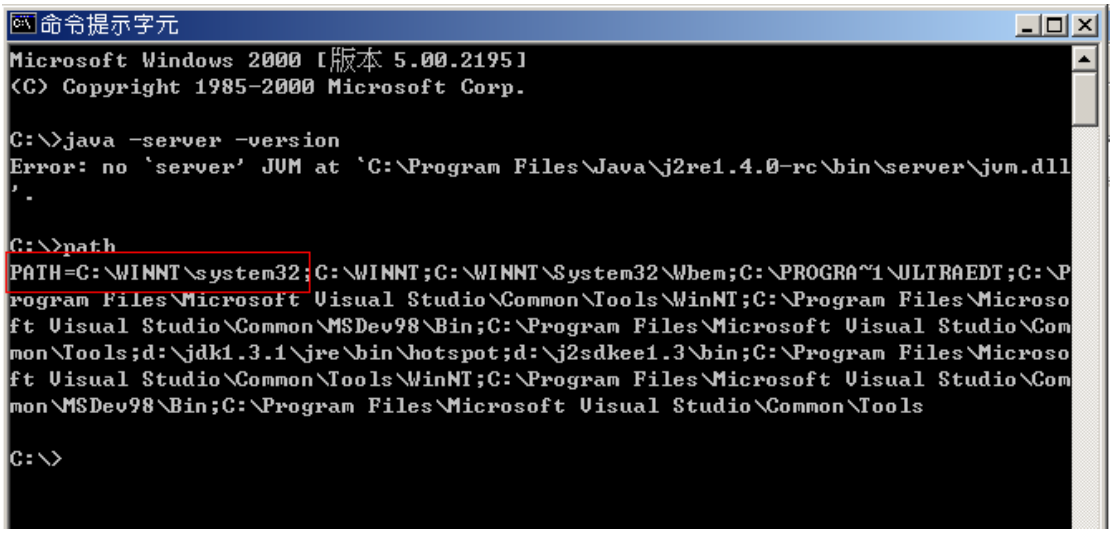

**圖:真正的 Java 虛擬機器之所在(hotspot server)**




(註:如果您設定 path=d:\jdk1.3.1\bin ,那麼就應該開啟 d:\jdk1.3.1\jre\bin,您將看到 classic、hotspot、與 server 三個子目錄,裡頭都擁有 jvm.dll)

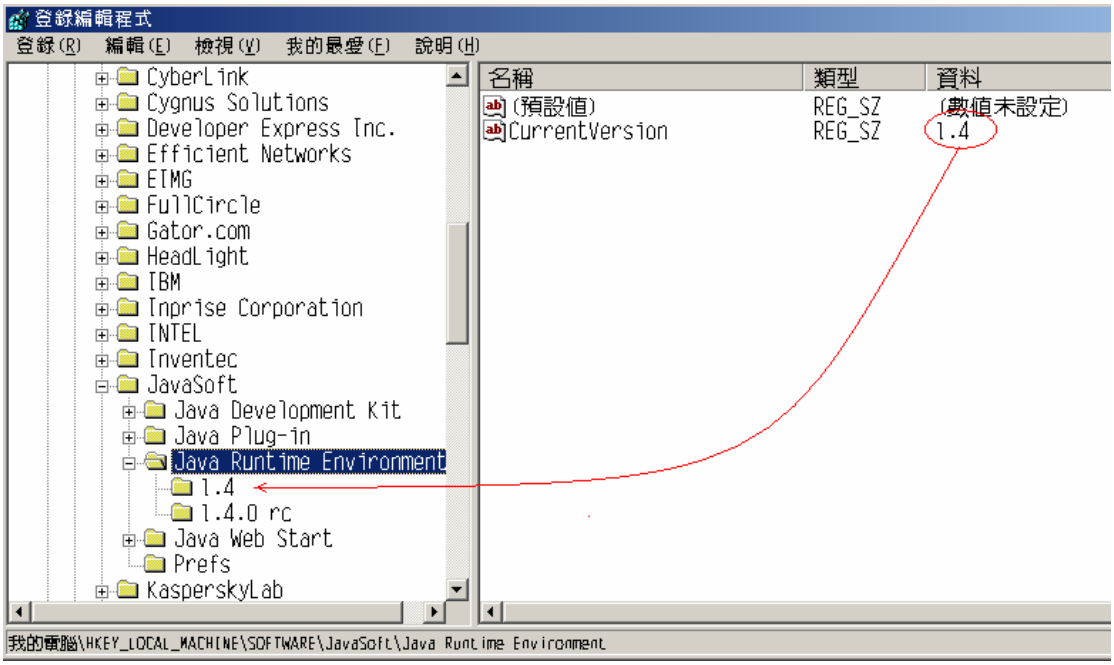
當您輸入的指令為 java –server 時,就會使 java.exe 自動選擇 d:\j2sdk1.4.0\jre\bin**server** 這個目錄底下的 JVM;而輸入 java –classic 時,就會使 java.exe 自動選擇 d:\jdk1.3.1\jre\bin**classic** 這個目錄底下的 JVM。從這裡我們就可以解釋之前為何設定了 path=c:\jdk1.3.1\bin 之後,會在畫面中突然出現兩個之前不曾出現的選項,也可以說明為何設定了 path=d:\j2sdk1.4.0\bin 之後,java –server 的指令會從原本的錯誤訊息變成可以正常使用。

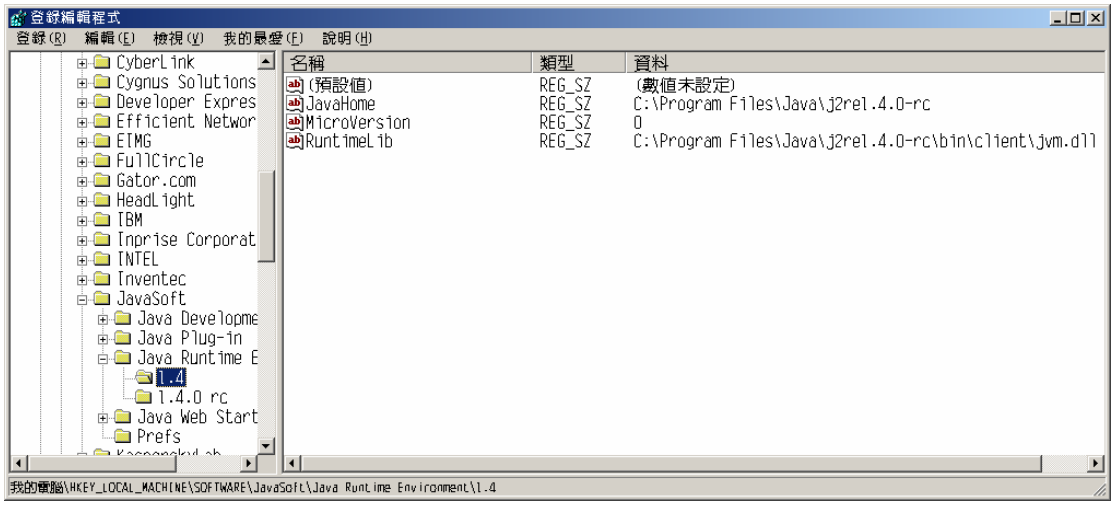
至於為何沒有設定 PATH 環境變數前,會出現錯誤訊息,這是因為系統預設的 PATH 環境變數內容如下:



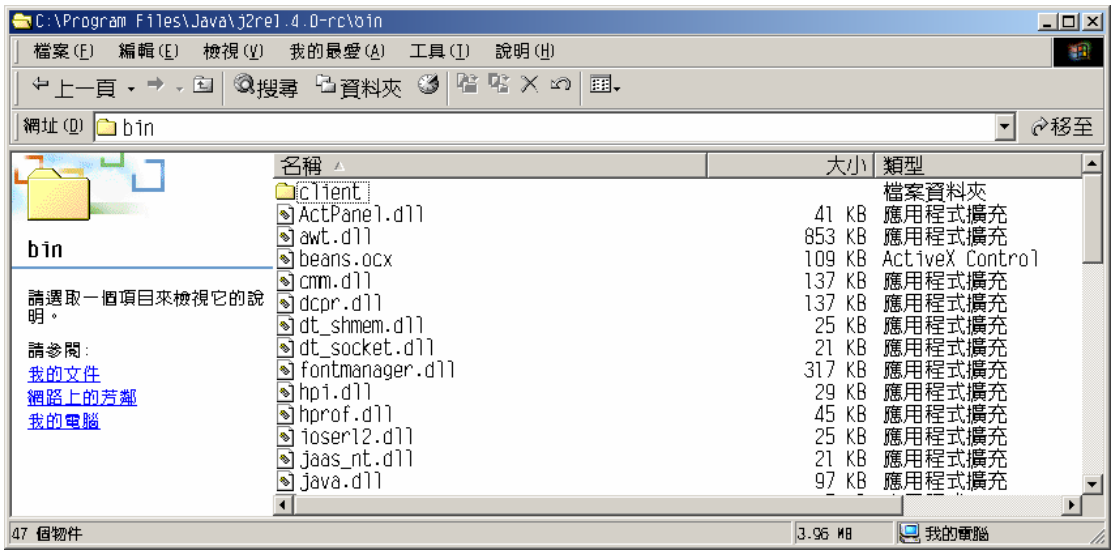

所以 java.exe 在 c:\winnt\system32 底下找不到 JRE 目錄，在 c:\winnt 也找不到 JRE 目錄的情況下，根據下一個邏輯，就是去查詢 Windows Registry，如下圖所示:

**圖:Windows Registry**

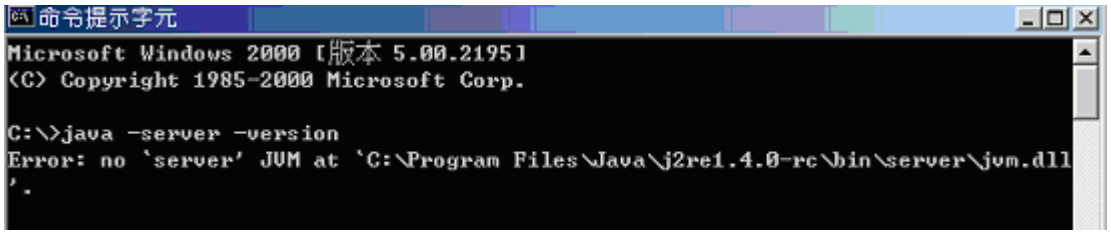





因為在 Windows Registry 之中查詢到 JRE 所在位置為 C:\Programm Foles\Java\j2re1.4.0-rc 。開啟該目錄下的 bin 子目錄卻只有看到 client 子目錄，卻沒有看到 server 子目錄:



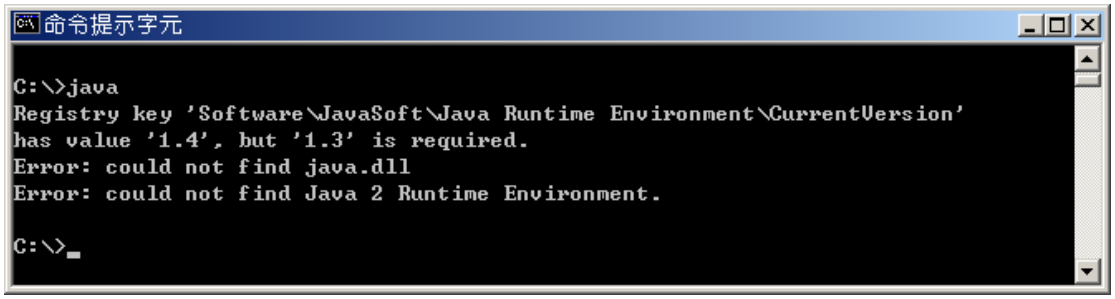

這就是之所以出現錯誤訊息:




的原因，因為 java.exe 根本無法在 JRE 所在目錄下的 bin 子目錄之中找到名為 server 的子目錄。同樣的道理，請您自行驗證使用 JDK 1.3.x 時螢幕上執行 java.exe 的結果之成因，您將發現 Registry 之中所設定 1.3.x 版的 JRE 之路徑下的 bin 子目錄，一樣只有 client 目錄。

其實，在 java.exe 找到 JRE 之後，還有一個驗證版本的程序，也就是 java.exe 和 JRE 兩者的版本要一致才能繼續執行。所以假如陰錯陽差，您執行

到 1.3.x 版 JDK 附的 java.exe，而此 java.exe 卻找到版本號碼為 1.4.x 的 JRE，就會發生底下結果:




上面林林總總解釋了那麼多，只想告訴大家，當您在開發 Java 程式或是執行 Java 程式的時候，一定要記得兩件事:

1. 那一個 java.exe 被執行。
2. java.exe 找到哪一套 JRE。

只要這兩件事都確定了，就知道問題發生的來龍去脈，也可以很容易地解決很多貌似靈異的怪問題。

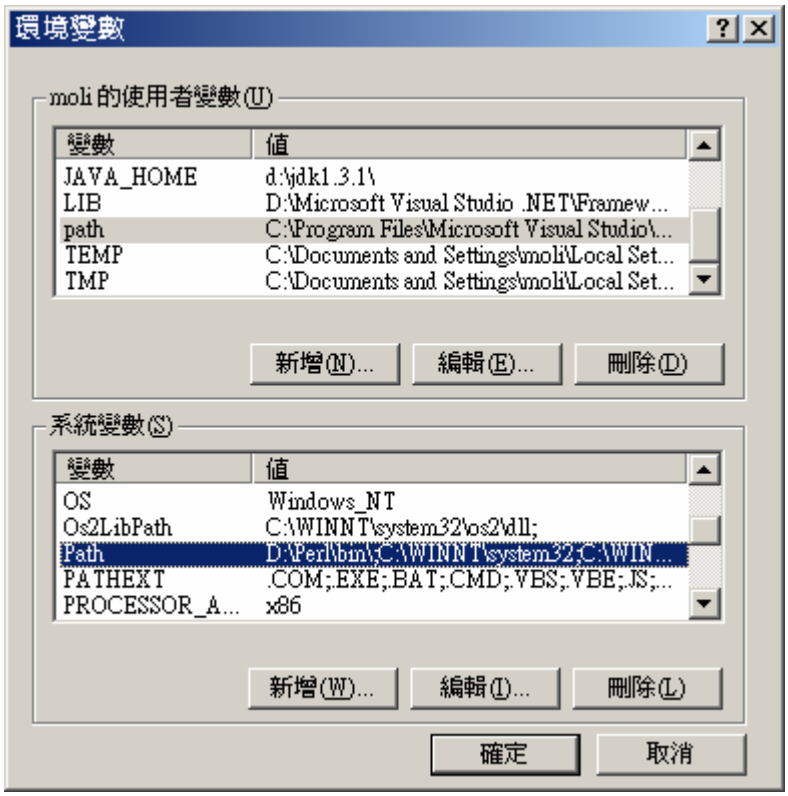
## 常見的錯誤

前面已經拿 JRE 與 PC 做了個比喻，JRE 對 Java 應用程式來說，就是一台獨立的電腦，而您的電腦上會有好幾套不同的 JRE，因此對 Java 應用程式來說，就是有許多不同的「虛擬電腦」。我們可以想像一種情況:您有兩台電腦，一台是 PC，一台是 Notebook，如果您在 Notebook 上安裝了 DirectX 函式庫，那麼 PC 上仍然無法執行需要 DirectX 的遊戲，您一定會在 PC 上重新安裝 DirectX。再舉個例子，您在 PC 上設定好作業環境(如日期設定、密碼設定、安全設定)之後，您一定不會期待 Notebook 上也有這些設定，您必須自己動手設定您的 Notebook 才行。這個道理很簡單 - 因為這兩台電腦根本就是兩個不同獨立的個體。

對 Java 應用程式來說，每個 JRE 都是獨立不相干的個體。舉凡函式庫、安全設定等與特定 JRE 相關聯的特性，如果您設定的是在 A 處的 JRE，但是執行時 Java 應用程式卻是在 B 處的 JRE 之中執行，那麼您的設定就會完全沒有作用。

所以，您必須清楚地知道哪一個 java.exe 被執行，以及 java.exe 到底選用到哪個 JRE，這些都是非常重要的細節。底下我們示範一個一般人幾乎不太去注意的小細節:

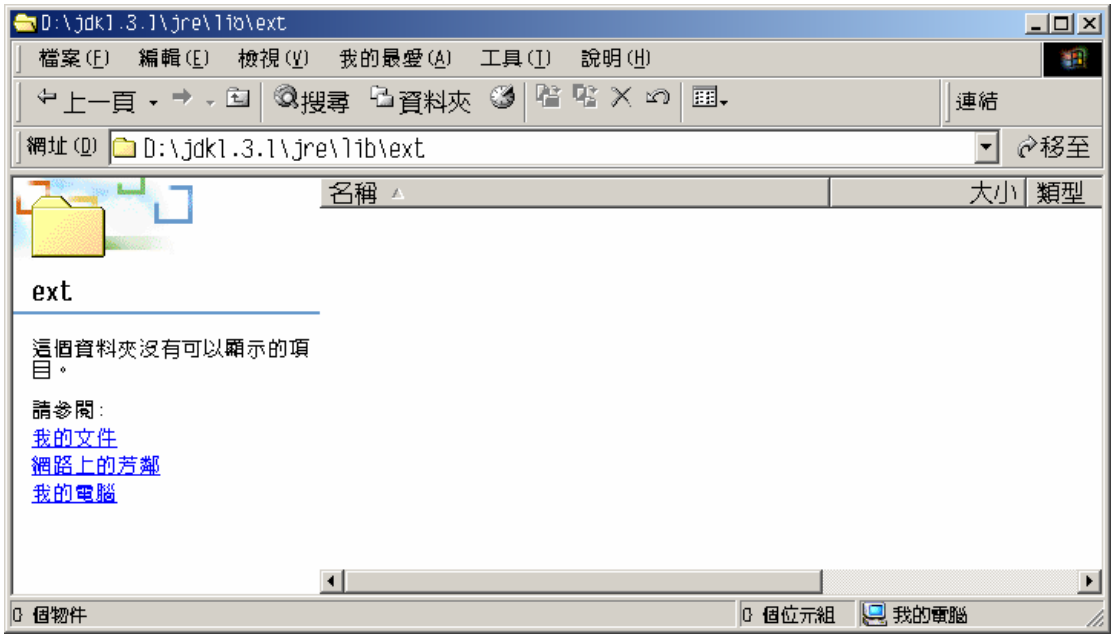
學習 Java 的朋友，一定會發現書上會希望您設定 PATH 環境變數，一旦 PATH 環境變數指向**<jdk 安裝目錄>\bin** 底下，您就可以在任何目錄下執行 java.exe。如果您使用的是 Windows NT/2000/XP 作業系統，設定方式如下:



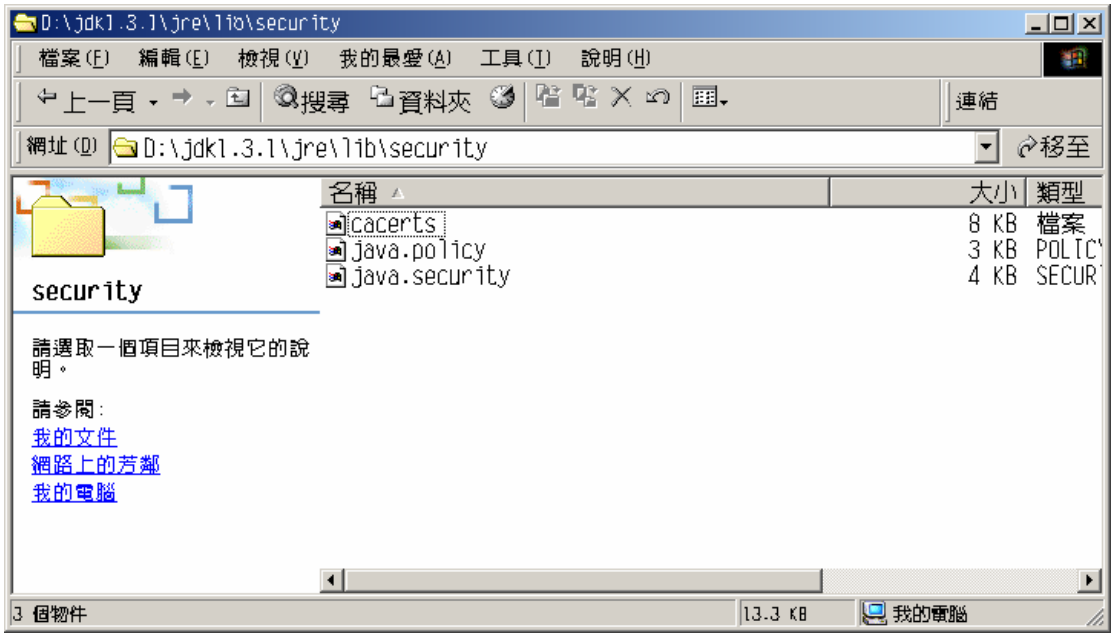

如上圖所示，上面是設定使用者變數，下面是設定系統變數。通常一般人會設定使用者變數如下:


注意，假設筆者的 JDK 1.3.1 安裝在 D 磁碟機，所以會在使用者變數的最開頭加上 **d:\jdk1.3.1\bin**。設定完成之後，就開始了您的 Java 程式設計學習之旅。

不過有一天，您學習的越來越深入時，您的生活會開始和 JRE 產生關聯。比方說您會學習不同的 Java 函式庫，函式庫的說明文件會希望您把這些函式庫(通常是一堆 JAR 檔)放到**<JRE 所在目錄>\lib\ext** 底下，通常您會選擇放到**<jdk 安裝目錄>\jre\lib\ext** 底下:



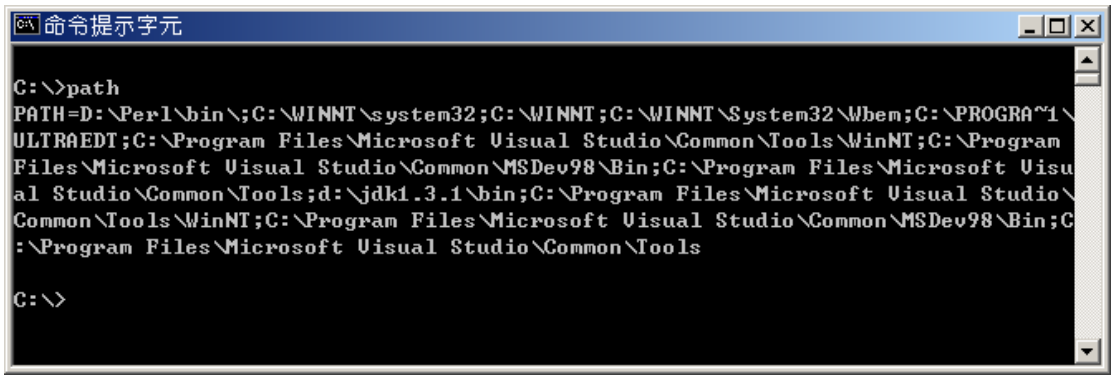

此外，您也有可能會學習 Java 的安全機制，與 Java 相關的安全設定檔都放置於**<JRE 所在目錄>\lib\security** 底下，改變這個目錄下的檔案設定，會可以改變 JVM 在安全上的設定，通常您也會選擇修改**<jdk 安裝目錄>\jre\lib\security** 目錄底下的檔案:




接下來，當您執行 Java 程式時，就會發現您的安全設定根本不會發生作用；而叫用了其他 Java 函式庫的程式，在編譯階段沒有問題，可是卻無法執行(螢幕

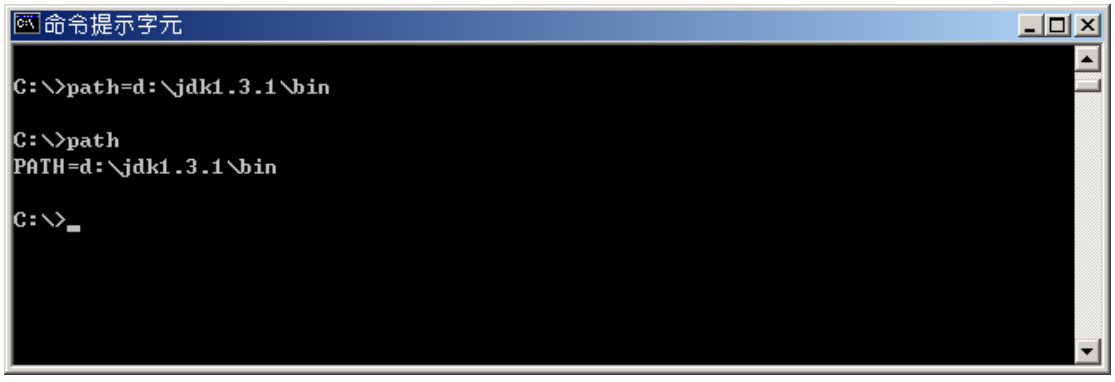
上會顯示 ClassNotFoundException)。

一般初學者無法理解問題的發生原因，而學習 Java 已久的老鳥也可能不清楚這個問題之所以產生的真正原因。之所以會發生此怪異情形，我們必須從命令提示模式下來看:

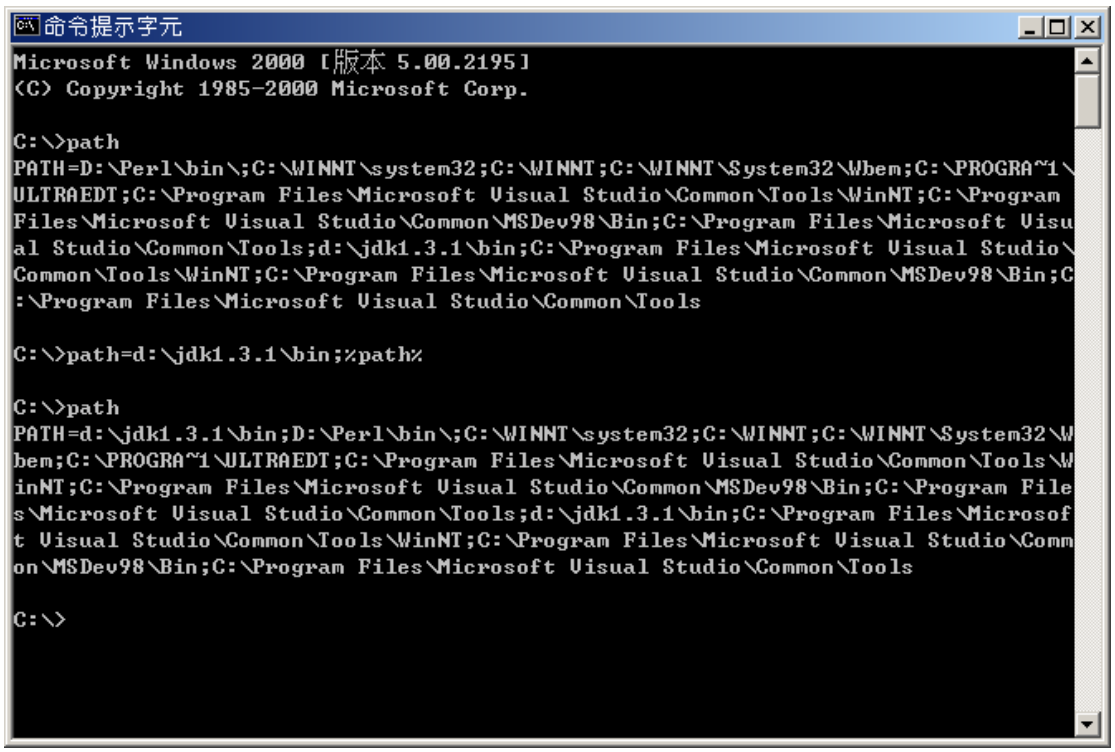



就我們之前對環境變數的設定來說，在命令提示模式下看起來如此(系統變數在前，使用者變數在後)，我們前面還曾經要大家去搜尋您的電腦中到底有幾個 java.exe(還記得嗎? C:\winnt\system32 與 d:\jdk1.3.1\bin 底下都曾出現 java.exe)，所以當您編譯 Java 應用程式時，由於只有 d:\jdk1.3.1\bin 底下可以找到 javac.exe，而 javac.exe 又會自動使用 JDK 所在目錄下的那一套 JRE，因此很自然您的函式庫如果拷貝到**<jdk 安裝目錄>\jre\lib\ext** 底下，很自然地在編譯時 JVM 會找到函式庫，所以編譯不會發生任何問題。但是執行時就有差了，因為 PATH 環境變數的關係，所以您打 java XXX 的時候，會優先執行 c:\winnt\system32 底下的那個 java.exe，也因此很自然地會去使用 c:\program files 目錄下的那一套 JRE，因此要讓您的 Java 應用程式可以正常執行，您還必須將那些函式庫拷貝到 c:\program files 目錄下的那一套 JRE 的 lib\ext 目錄之下。Java 安全設定之所以沒有作用，也是基於一樣的道理，如果以上述的情境來說，您就必須修改 c:\program files 目錄下的那一套 JRE 的 lib\security 目錄中的設定檔才行。

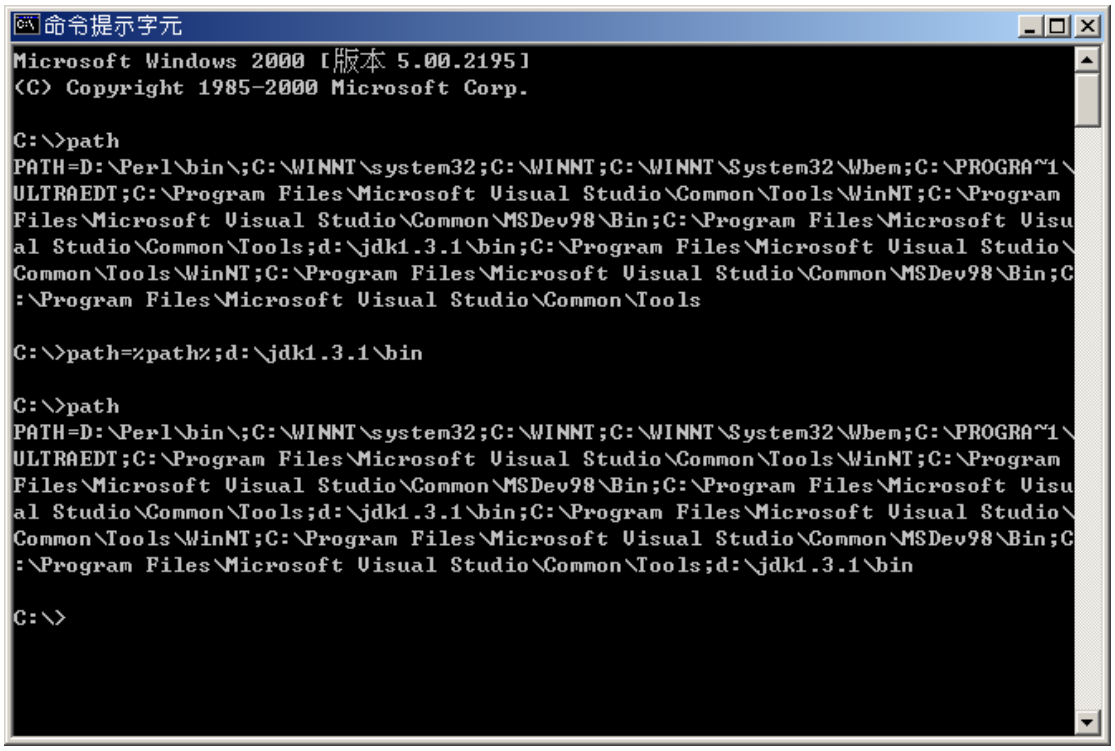
如果您是每次都在命令提示模式下手動設定 PATH 環境變數如下:




就不會發生上述問題，因為不管是 javac.exe 或 java.exe，都會執行 JDK 所在目錄下的，也因此找到的都是 JDK 安裝目錄下的那套 JRE，自然就沒有上述問題。當然，我們可以類推，如果您設定的方式為:




就不會發生奇怪的問題，而如果您設定的方式為



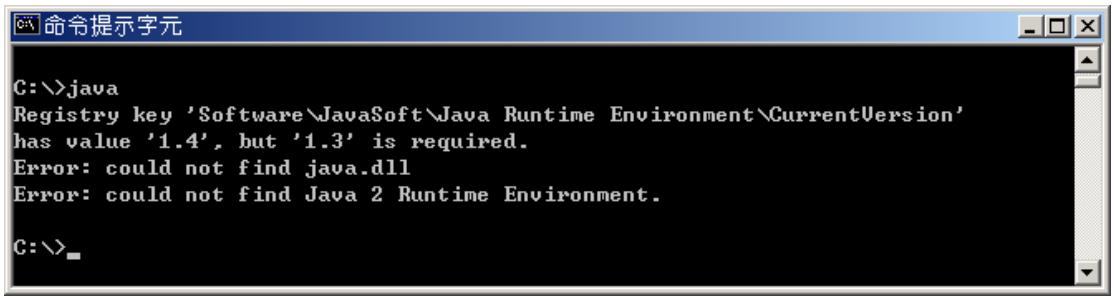

則會發生上述的奇怪問題。

## 總結

在本文中，筆者為大家介紹 JDK、JRE、以及 JVM 三者之間的關係。依照

這三者的關係，筆者為大家解釋了很多學習 Java 時遇到的靈異現象。只要知道事情的來龍去脈，靈異就不再是靈異，而且我們還可以去玩弄這些運作機制。

在看本章之後，如果要證明您已經了解本章內容，請容許筆者出個小題目幫您做個測試。還記得底下這個錯誤訊息嗎?




請試著「刻意地」造出此錯誤訊息。如果您成功了，那麼您已經開始進入玩弄 JDK、JRE、以及 JVM 的層次了。