Appendice A

Installazione dell'SDK

GIOVANNI PULITI

Scelta del giusto SDK

Per poter lavorare con applicazioni Java o crearne di nuove, il programmatore deve poter disporre di un ambiente di sviluppo e di esecuzione compatibile con lo standard 100% Pure Java. Sun da sempre rilascia un kit di sviluppo che contiene tutti gli strumenti necessari per la compilazione ed esecuzione di applicazioni Java. Tale kit è comunemente noto come Java Development Kit (JDK): nel corso del tempo sono state rilasciate le versioni 1.0, 1.1, 1.2, 1.3 e 1.4. Attualmente l'ultima versione rilasciata è la 1.4, mentre si annuncia un prossimo JDK 1.5. Il JDK comprende una Java Virtual Machine (JVM), invocabile con il comando `java`, un compilatore (comando `javac`), un debugger (`jdbg`), un interprete per le applet (`appletviewer`) e altro ancora.

A partire dalla versione 1.2, Sun ha introdotto una nomenclatura differente per le varie versioni del kit di sviluppo. In quel momento nasceva infatti Java 2, a indicare la raggiunta maturità del linguaggio e della piattaforma. Pur mantenendo la completa compatibilità con il passato, Java 2 ha introdotto importanti miglioramenti, quali una maggiore stabilità e sicurezza, migliori performance e l'ottimizzazione dell'uso della memoria.

Con Java 2 nasce il concetto di SDK: non più un Java Development Kit ma un Software Development Kit. Il linguaggio Java può essere finalmente considerato un potente strumento general purpose.

La notazione di JDK non è stata eliminata: il JDK è formalmente una release dell'SDK Java 2.

Con Java 2, per organizzare e raccogliere al meglio le diverse tecnologie che costituiscono ormai la piattaforma, Sun ha suddiviso l'SDK in tre grandi categorie:

- Java 2 Standard Edition (J2SE): questa versione contiene la JVM standard più tutte le librerie necessarie per lo sviluppo della maggior parte delle applicazioni Java.

- Java 2 Enterprise Edition (J2EE): contiene in genere le API enterprise come EJB, JDBC 2.0, Servlet ecc. La JVM normalmente è la stessa, quindi lavorare direttamente con l'SDK in boundle spesso non è molto utile: è molto meglio partire dalla versione J2SE e aggiungere l'ultima versione delle API EE, a seconda delle proprie esigenze.

- Java 2 Micro Edition (J2ME): Java è nato come linguaggio portabile in grado di essere eseguito con ogni tipo di dispositivo. La J2ME include una JVM e un set di API e librerie appositamente limitate, per poter essere eseguite su piccoli dispositivi embedded, telefoni cellulari ed altro ancora. Questa configurazione deve essere scelta solo se si vogliono scrivere applicazioni per questo genere di dispositivi.

La procedura di installazione è in genere molto semplice, anche se sono necessarie alcuni piccoli accorgimenti per permettere un corretto funzionamento della JVM e dei programmi Java. Si limiterà l'attenzione alla distribuzione J2SE. Per chi fosse interessato a scrivere programmi J2EE non vi sono particolari operazioni aggiuntive da svolgere. Per la J2ME, invece, il processo è del tutto analogo, anche se si deve seguire un procedimento particolare a seconda del dispositivo scelto e della versione utilizzata.

I file per l'installazione possono essere trovati direttamente sul sito di Sun, come indicato in [SDK]. Altri produttori rilasciano JVM per piattaforme particolari (molto note e apprezzate sono quelle di IBM). Per la scelta di JVM diverse da quelle prodotte da Sun si possono seguire le indicazioni della casa produttrice o del particolare sistema operativo utilizzato.

Chi non fosse interessato a sviluppare applicazioni Java, ma solo a eseguire applicazioni già finite, potrà scaricare al posto dell'SDK il Java Runtime Environment (JRE) , che in genere segue le stesse edizioni e release dell'SDK. Non sempre il JRE è sufficiente: per esempio, se si volessero eseguire applicazioni JSP già pronte, potrebbe essere necessario mettere ugualmente a disposizione di Tomcat (o di un altro servlet-JSP engine) un compilatore Java, indispensabile per la compilazione delle pagine JSP.

Installazione su Windows

In ambiente Windows, in genere, il file di installazione è un eseguibile autoinstallante, che guida l'utente nelle varie fasi della procedura.

Non ci sono particolari aspetti da tenere in considerazione, a parte la directory di installazione e le variabili d'ambiente da configurare. Per la prima questione, a volte l'installazione nella directory "program files" può causare problemi di esecuzione ad alcuni applicativi Java che utilizzano la JVM di sistema (per esempio Tomcat o JBoss). Per questo, si consiglia di installare in una directory con un nome unico e senza spazi o altri caratteri speciali ("c:\programs", "c:\programmi" o semplicemente "c:\java").

Per poter funzionare, una qualsiasi applicazione Java deve sapere dove è installato il JDK, e quindi conoscere il path dell'eseguibile java (l'interprete), di javac (il compilatore usato da Tomcat per compilare le pagine JSP) e di altri programmi inclusi nel JDK.

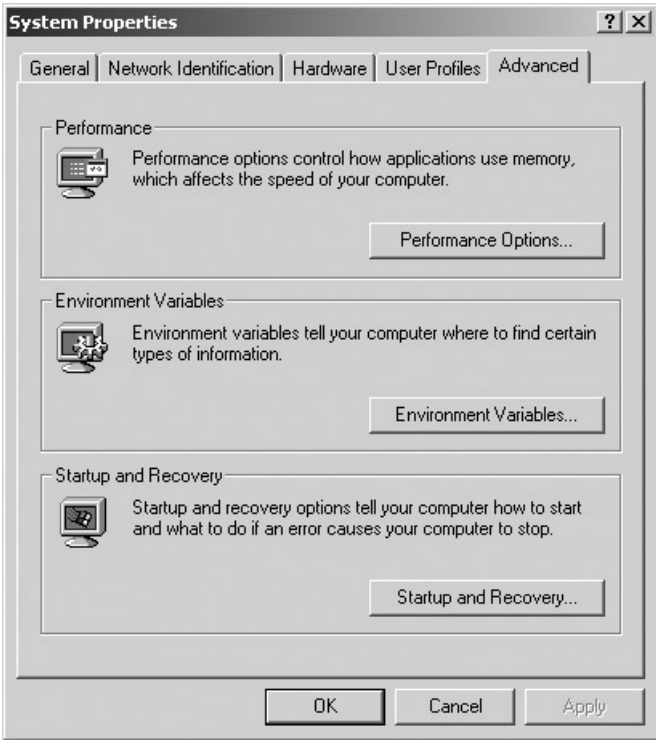
Inoltre, un programma Java deve anche poter ricavare la variabile d'ambiente CLASSPATH, all'interno della quale dovranno essere inseriti i riferimenti ai vari package utilizzati (directory scompattate, file .jar o .zip). Al momento dell'installazione, il classpath viene automaticamente impostato in modo da contenere le librerie di base del Java SDK (in genere, nella sottodirectory jre/lib, o semplicemente lib).

A partire dal JDK 1.1 è invalsa l'abitudine di utilizzare la variabile JAVA_HOME, che deve puntare alla directory di installazione del JDK. Di conseguenza, il path di sistema dovrà essere impostato in modo che punti alla directory %JAVA_HOME%\bin.

Di norma, queste impostazioni sono effettuate in modo automatico dal programma di installazione, ma possono essere facilmente modificate o impostate ex-novo tramite il pannello di controllo di Windows.

Per esempio, aprendo la finestra per l'impostazione delle variabili d'ambiente dal pannello di controllo...

Figura A.1 – *In Windows il pannello di controllo permette di impostare le diverse variabili d'ambiente.*



...si può procedere a inserire sia la variabile JAVA_HOME (in questo caso c:\programs\jdk1.4) sia la directory con gli eseguibili nel path (%JAVA_HOME%\bin).

Figura A.2 – *Come impostare la variabile JAVA_HOME in modo che punti alla directory di installazione del JDK.*

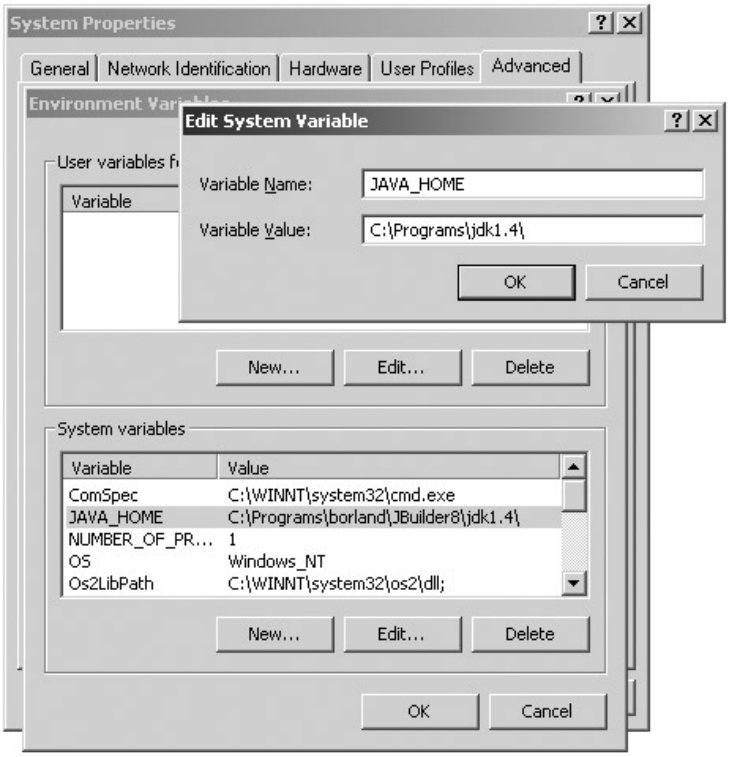
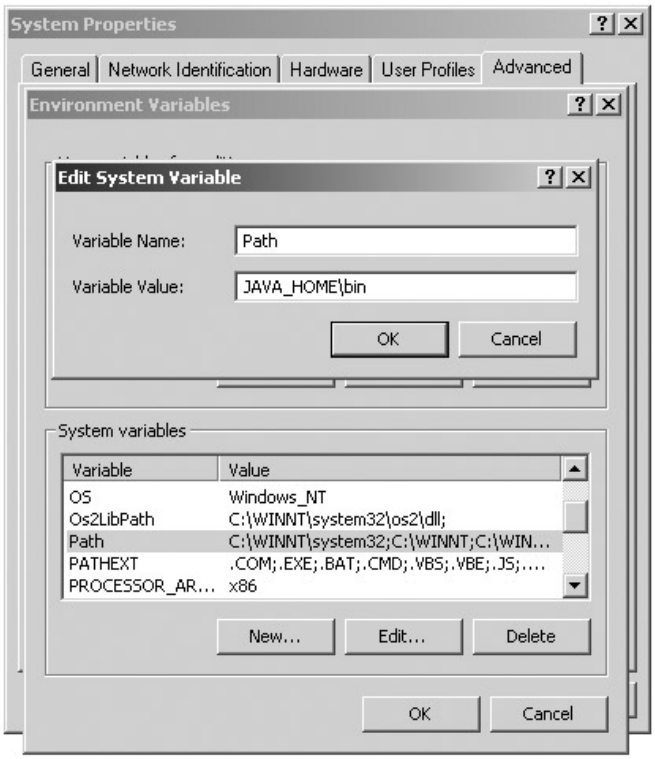


Figura A.3 – *Come impostare il path in modo che includa la dir* JAVA_HOME\bin.



Se tutto è stato fatto come si deve, aprendo una console DOS si può verificare la correttezza delle impostazioni inserite.

Per esempio, per conoscere il contenuto della variabile JAVA_HOME si potrà scrivere:

```
C:\>echo %JAVA_HOME%
C:\programs\jdk1.4
```

Il comando path, invece, mostrerà Tra le altre cose:

```
C:\> path
PATH=…….C:\programs\jdk1.4\bin
```

A questo punto, si può provare a eseguire la JVM con il comando:

```
C:\>java -version
```

L'opzione –version permette di conoscere la versione della JVM installata. In questo caso, il comando restituisce il seguente output:

```
java version "1.4.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1-b21)
Java HotSpot(TM) Client VM (build 1.4.1-b21, mixed mode)
```

se si usa un sistema basato su Windows 95-98 o ME, l'impostazione delle variabili d'ambiente può essere fatta tramite il file autoexec.bat. In questo caso, con un'istruzione del tipo:

```
set JAVA_HOME="…."
```

Si potrà definire la variabile in modo che punti alla directory indicata. Si tenga presente che tali sistemi operativi offrono un supporto ridotto per lo sviluppo e l'esecuzione di applicazioni che fanno uso dei protocolli di rete (socket TCP, database ecc...) o server side (servlet, web application, EJB, per esempio). Si consiglia pertanto di utilizzare le versioni più evolute (NT, 2000, XP), che supportano in modo più corretto e completo lo strato di network TCP/IP e offrono maggiori servizi.

Installazione su Linux

Per l'installazione su Linux, la procedura è molto simile a quella per Windows: si deve scaricare un file di installazione e installarlo. Per quest'ultimo aspetto si può utilizzare un rpm autoinstallante o un file eseguibile con estensione .bin.

Per esempio, se j2sdk-1.4.2-nb-3.5-bin-linux.bin è il file installante scaricato dal sito Sun, per prima cosa lo si renda eseguibile con il comando:

```
chmod o+x j2sdk-1.4.2-nb-3.5-bin-linux-i586.bin
```

quindi lo si mandi in esecuzione tramite:

```
./j2sdk-1.4.2-nb-3.5-bin-linux-i586.bin
```

per eseguire l'installazione. Di norma, questo porterà all'installazione dell'SDK in una directory il cui nome segue lo schema usr/java/jdk-<version-number>.

Questo significa che dovranno essere modificate di conseguenza le variabili JAVA_HOME e PATH, intervenendo sui file di profilo .bashrc o .bash_properties (a seconda del tipo di shell usata) dell'utente che dovrà usare Java:

```
JAVA_HOME=/usr/java/jdk1.4.1/
export JAVA_HOME
PATH=$JAVA_HOME/bin:$PATH
export PATH
```

Nel caso in cui un'applicazione debba far uso di altri package oltre a quelli di base del Java SDK, come un parser XML Xerces (contenuto in genere in un file xerces.jar), il package Java Mail, la Servlet API o altro ancora, si dovrà aggiungere manualmente al classpath il contenuto di tali librerie. Questo può essere fatto in due modi.

Il primo sistema consiste nell'aggiungere tali librerie al classpath di sistema, tramite il pannello di controllo di Windows o mediante l'impostazione ed esportazione di una variabile globale su Linux. In questo caso si potrà essere sicuri che tutte le applicazioni che dovranno utilizzare un parser Xerces o JavaMail potranno funzionare correttamente senza ulteriori impostazioni.

Attualmente, però, lo scenario Java è molto complesso, quindi un'impostazione globale difficilmente si adatta a tutte le applicazioni: in un caso potrebbe essere necessaria la versione 1.0 di Xerces, mentre un'altra applicazione potrebbe funzionare solo con la 1.2. Per questo motivo, in genere, si preferisce impostare un classpath personalizzato per ogni applicazione, passando tale configurazione alla JVM con il flag –classpath o –cp. Per esempio, in Windows si potrebbe scrivere:

```
set MY_CP=c:\programs\mokabyte\mypackages.jar
java –cp %MY_CP% com.mokabyte.mokacode.TestClasspathApp
```

Dove TestClasspathApp potrebbe essere un'applicazione che abbia bisogno di una serie di classi e interfacce contenute in mypackages.jar.

In questo modo si potranno costruire tutti i classpath personalizzati, concatenando file e directory di vario tipo.

In ambito J2EE le cose si complicano: entrano infatti in gioco il tipo di applicazione e le regole di caricamento del classloader utilizzato. Per questi aspetti, che comunque riguardano il programmatore esperto, si rimanda alla documentazione del prodotto utilizzato, e si consiglia l'adeguamento alle varie convenzioni imposte dalla specifica Java.

Bibliografia

[SUN] – Sito web ufficiale di Sun dedicato a Java: http://java.sun.com

[SDK] – Sito web di Sun per il download dell'SDK nelle versioni per le varie piattaforme: http://java.sun.com/downloads

[JIBM] – Sito web IBM dedicato a Java: http://www.ibm.com/java

[xIBM] – "IBM Developer Kit for Linux: Overview": http://www-106.ibm.com/developerworks/java/jdk/linux140/?dwzone=java