Java: preparazione

Java è un linguaggio di programmazione realizzato da Sun Microsystems, utilizzato originariamente per l'inserzione di programmi all'interno di pagine HTML (applet), un po' come si fa con le immagini. Per questo motivo, il risultato consueto della compilazione di un sorgente Java è una codifica intermedia, indipendente dalla piattaforma, che deve poi essere interpretata localmente dal navigatore o da un altro programma indipendente. Tuttavia, nel tempo sono stati sviluppati anche compilatori alternativi, che producono un programma eseguibile tradizionale (dipendente dalla piattaforma hardware-software).

Per programmare in Java occorre un compilatore, generalmente noto come '**javac**', che sia in grado di generare il formato binario Java, il cosiddetto Java bytecode. Il file che si ottiene non è propriamente un eseguibile, in quanto necessita di un interprete che generalmente è il programma '**java**'.

Esiste una versione ufficiale di questi strumenti, definita JDK (*Java development kit*), e altre versioni indipendenti, come per esempio Kaffe.

Nel capitolo viene descritto in particolare come utilizzare Kaffe. Alla fine del capitolo si trova la descrizione dell'installazione e della configurazione di JDK originale, oltre a una sezione sull'uso di GCJ per la compilazione di sorgenti o binari Java nel formato eseguibile adatto alla propria architettura.

Kaffe

«

Kaffe [1] è un compilatore di sorgenti Java e un interprete di compilati in formato Java (Java bytecode). Attualmente, si tratta di un pacchetto standard delle distribuzioni GNU, per cui non ci dovrebbero essere problemi nella sua installazione. Attualmente, assieme al compilatore e all'interprete, dovrebbero essere disponibili anche le ***classi***, ovvero le librerie Java.

Classi

«

Le classi di Kaffe, che ormai accompagnano questo applicativo, dovrebbero essere contenute in un solo file compresso, che deve rimanere tale. Potrebbe trattarsi di '`/usr/share/kaffe/Klasses.jar`'.

Se si installa Kaffe autonomamente, senza affidarsi a un pacchetto già predisposto per la propria distribuzione GNU, potrebbe essere necessario definire alcune variabili di ambiente. Nell'esempio seguente si fa riferimento a uno script per una shell Bourne o derivata:

```
CLASSPATH=.:/usr/share/kaffe/Klasses.jar
KAFFEHOME=/usr/share/kaffe
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib
export CLASSPATH
export KAFFEHOME
export LD_LIBRARY_PATH
```

Se Kaffe fosse stato installato a partire dalla directory '`/usr/local/`', si dovrebbe usare la definizione seguente:

```
CLASSPATH=.:/usr/local/share/kaffe/Klasses.jar
KAFFEHOME=/usr/local/share/kaffe
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib
export CLASSPATH
export KAFFEHOME
export LD_LIBRARY_PATH
```

Merita un po' di attenzione la variabile '**`LD_LIBRARY_PATH`**' che potrebbe essere utilizzata anche da altri programmi. '**`LD_LIBRARY_PATH`**' deve contenere i percorsi in cui si trovano i file di libreria; se il proprio sistema utilizza applicazioni che collocano le proprie librerie all'interno di directory inconsuete, queste devono essere aggiunte all'elenco. Segue un esempio esplicativo:

```
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib:/opt/mio_prog/lib:/opt/tuo_prog/lib
```

Compilazione

Per verificare che la compilazione funzioni correttamente, basta preparare il solito programma banale che visualizza un messaggio attraverso lo standard output e poi termina:

```
class CiaoMondoApp
{
    public static void main (String[] args)
      {
        System.out.println ("Ciao Mondo!");
      }
}
```

Il file deve essere salvato con il nome '`CiaoMondoApp.java`'. Kaffe, tra le altre cose, fornisce un collegamento simbolico, denominato '**`javac`**', attraverso cui avviare la compilazione. Così la compilazione avviene nello stesso modo degli strumenti JDK originali:

```
$ javac CiaoMondoApp.java [Invio]
```

Se la sintassi del sorgente Java è corretta, si ottiene un file in formato binario Java, denominato '`CiaoMondoApp.class`'.

Esecuzione

Per eseguire il binario Java generato, ovvero il file '`.class`', occorre un interprete. In questo senso, il binario Java non ha bisogno necessariamente dei permessi di esecuzione, perché viene solo letto dall'interprete.

```
$ kaffe CiaoMondoApp [Invio]

  Ciao Mondo!
```

Come si può osservare dalla riga di comando, il file binario Java deve essere indicato senza l'estensione, che di conseguenza è obbligatoriamente '`.class`'. Kaffe si compone anche dello script '**`java`**', il cui scopo è quello di rendere il comando di interpretazione conforme al JDK; in pratica, '**`java`**' si limita ad avviare il comando '**`kaffe`**'.

`$` **`java CiaoMondoApp`** [*Invio*]

Tuttavia, questo script potrebbe essere modificato in modo da permettere l'avvio di un eseguibile Java anche se è stato fornito il nome del file corrispondente, completo di estensione '`.class`'. L'esempio seguente rappresenta le modifiche che potrebbero essere apportate in tal senso:

```
#!/bin/sh
#
# /usr/bin/java

CLASSE=`/bin/basename $1 .class`
shift
kaffe $CLASSE $@
```

Kernel Linux

Come è noto, uno script viene interpretato automaticamente in base alla convenzione per cui la prima riga inizia con l'indicazione del programma adatto. Per esempio: '**`#!/bin/sh`**', '**`#!/bin/bash`**' e '**`#!/usr/bin/perl`**'. Con i binari Java ciò non è possibile, quindi, per ottenere l'avvio automatico dell'interprete '**`java`**', occorre che il kernel ne sia informato. Per la precisione, occorre attivare la funzionalità generica di riconoscimento dei binari (sezione 8.3.1); inoltre occorre accertarsi che la directory '`/proc/sys/fs/binfmt_misc/`' contenga i file '`register`' e '`status`'.

Se le cose non stanno così, è necessario innestare il file system '**binfmt_misc**':

```
# mount -t binfmt_misc none /proc/sys/fs/binfmt_misc[Invio]
```

Una volta che sono disponibili i file virtuali '`register`' e '`status`', per attivare la funzionalità occorre intervenire con il comando seguente:

```
# echo 1 > /proc/sys/fs/binfmt_misc/status[Invio]
```

Per disattivarla, basta utilizzare il valore zero.

```
# echo 0 > /proc/sys/fs/binfmt_misc/status[Invio]
```

Quando tutto è in ordine per la gestione dei binari eterogenei, si può definire quali file devono essere riconosciuti e quali interpreti devono essere avviati di conseguenza. Nel caso dei binari Java normali, si tratta di eseguire il comando seguente (il percorso dell'interprete, '`/usr/bin/java`' può essere cambiato a seconda delle proprie necessità).

```
# echo ':Java:M::\xca\xfe\xba\xbe::/usr/bin/java:'  ↩
↪> /proc/sys/fs/binfmt_misc/register[Invio]
```

In alternativa, se si è sicuri dell'estensione '`.class`', si può utilizzare il comando seguente:

```
# echo ':Java:E::class::/usr/bin/java:'  ↩
↪> /proc/sys/fs/binfmt_misc/register[Invio]
```

Per verificare che la definizione sia stata recepita correttamente dal kernel, si può leggere il contenuto del file virtuale '`/proc/sys/fs/binfmt_misc/Java`', creato a seguito di uno dei due comandi mostrati sopra.

Quando il kernel è predisposto nel modo appena visto, si possono rendere eseguibili i file binari Java; così, quando si tenta di avviarli, il kernel stesso avvia invece il comando seguente:

```
java file_binario_java argomenti
```

Lo svantaggio di questo sistema sta nel fatto che il nome del file binario Java viene indicato con tutta l'estensione, cosa che normalmente crea dei problemi, sia a Kaffe che al JDK. Per questo, conviene che '`/usr/bin/java`' sia uno script predisposto per risolvere il problema, come già mostrato nella sezione precedente.

Se invece di usare Kaffe si usa il JDK originale, conviene modificare il nome dell'interprete Java, per esempio in '**`java1`**', realizzando poi un file script analogo a quello già visto.

```
#!/bin/sh
#
# /usr/bin/java

CLASSE=`/bin/basename $1 .class`
shift
java1 $CLASSE $@
```

C'è però una cosa che occorre tenere a mente. Con GNU/Linux, così come con altri sistemi, non è possibile avviare un eseguibile se il nome non viene indicato per esteso. In pratica, non è pensabile che succeda quanto accade in Dos in cui i file che finiscono per '`.COM`' o '`.EXE`' sono avviati semplicemente nominandoli senza estensione.

Per chi ha usato GNU/Linux da un po' di tempo ciò dovrebbe essere logico, ma con Java si rischia ancora di essere ingannati: il fatto che, sia l'interprete '**`java`**' originale, sia '**`kaffe`**', vogliano il nome dell'eseguibile Java senza l'estensione '`.class`', non deve fa-

re supporre che ciò valga anche per il kernel. Per cui, se si avvia '`CiaoMondoApp.class`' nel modo seguente,

```
$ java CiaoMondoApp [Invio]
```

quando si vuole che sia il kernel a fare tutto questo in modo automatico, il comando diviene il seguente:

```
$ CiaoMondoApp.class [Invio]
```

Se si tentasse si eseguire il comando seguente, si otterrebbe una segnalazione di errore del tipo: '**`command not found`**'.

```
$ CiaoMondoApp [Invio]
```

Applet

«

Un'applet Java è un programma particolare che può essere incorporato in un documento HTML. Il meccanismo è simile all'inserzione di immagini; l'effetto è quello di un programma grafico che, invece di utilizzare una finestra si inserisce in un'area prestabilita del documento HTML. Un'applet Java non può quindi vivere da sola, richiede sempre l'abbinamento a una pagina HTML.

Il modo migliore per vedere il funzionamento di un programma del genere è attraverso l'utilizzo di un navigatore in grado di eseguire tali applet.

Verifica del funzionamento

«

Per verificare il funzionamento di un'applet si può provare il solito programma banale. In questo caso si comincia con la realizzazione di una pagina HTML che incorpori l'applet che si vuole realizzare.

```
<!-- CiaoMondo.html -->
<HTML>
<HEAD>
    <TITLE>La mia prima applet</TITLE>
</HEAD>
<BODY>
<OBJECT CODETYPE="application/java"
    CLASSID="java:CiaoMondo.class">
Applet Java
</OBJECT>
</BODY>
</HTML>
```

Come si vede, l'elemento '**OBJECT**' dichiara l'utilizzo dell'applet 'CiaoMondo.class'. Segue il sorgente dell'applet:

```
// CiaoMondo.java

import java.applet.Applet;
import java.awt.Graphics;

public class CiaoMondo extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("Ciao Mondo!", 50, 25);
    }
}
```

Si compila il sorgente 'CiaoMondo.java' nel solito modo, ottenendo il binario Java 'CiaoMondo.class'

```
$ javac CiaoMondo.java[Invio]
```

Quando si carica il file 'CiaoMondo.html' attraverso un navigatore adatto, incontrando l'elemento '**OBJECT**' che fa riferimento al binario Java '**CiaoMondo.class**', viene caricato il programma 'CiaoMondo.class' nell'area stabilita.

All'interno di quell'area, a partire dall'angolo superiore sinistro, vengono calcolate le coordinate (x=50, y=25) dell'istruzione

‘**`g.drawString("Ciao mondo!", 50, 25)`**’ vista nell’applet.

JDK

«

software non libero

JDK [2] è il pacchetto originale per la compilazione e l’esecuzione di applicativi Java. Viene distribuito in forma binaria, già compilata. Per ottenerlo, si può consultare *http://www.blackdown.org/* o eventualmente si può fare una ricerca attraverso *http://www.google.com* per i file contenenti la stringa ‘`linux-jdk`’ (si potrebbero trovare nomi come ‘`linux-jdk.1.1.3-v2.tar.gz`’). Se si desidera installare il JDK è importante verificare di non avere tracce di Kaffe.

Il JDK può essere installato a partire da qualunque punto del proprio file system. Qui viene proposta l’installazione a partire da ‘`/opt/`’.

Se nel proprio sistema non è presente, la si può creare, quindi al suo interno si può espandere il contenuto del pacchetto JDK. Si ottiene così la directory ‘`jdk`***versione***`/`’, per esempio ‘`jdk1.1.3/`’. Per motivi pratici è opportuno modificare il nome della directory, o creare un collegamento simbolico, in modo che vi si possa accedere utilizzando il nome ‘`/opt/java/`’.

Prima di poter funzionare, il JDK deve essere configurato attraverso delle variabili di ambiente opportune. Nell’esempio seguente si mostra un pezzo di script per una shell Bourne o derivata, in grado di predisporre le variabili necessarie:

```
PATH="/opt/java/bin:$PATH"
CLASSPATH=.:/opt/java/lib/classes.zip:/opt/java/lib/classes
JAVA_HOME=/opt/java
export PATH
export CLASSPATH
export JAVA_HOME
```

Per il funzionamento si può rivedere quanto già indicato per Kaffe. In questo caso, utilizzando il JDK originale, il compilatore è proprio '**javac**' e l'esecutore (o interprete) è '**java**'.

GCJ

GCJ [3] è un programma frontale per il controllo del compilatore GCC e di altri programmi accessori, il cui scopo è quello di compilare sorgenti Java.

La compilazione può avvenire a diversi livelli: da sorgenti Java ('`.java`') o da binari Java ('`.class`') si può arrivare a un file eseguibile per il proprio sistema operativo; in alternativa si possono semplicemente compilare dei sorgenti Java per generare i binari Java corrispondenti ('`.class`'). Semplificando le cose, si possono distinguere questi tre tipi di comandi per la compilazione:

- `gcj -C` ***file_sorgente_java*** ...

  per generare binari Java (file '`.class`');

- `gcj --main=`***classe_principale*** `-o` ***file_da_generare*** ***file_sorgente_java*** ...

  per generare un eseguibile a partire da dei sorgenti Java (file '`.java`');

- `gcj --main=`***classe_principale*** `-o` ***file_da_generare*** ***binario_java***...

  per generare un eseguibile a partire da binari Java (file '`.class`').

Supponendo di avere il solito esempio già visto in precedenza,

```
class CiaoMondoApp
{
    public static void main (String[] args)
    {
        System.out.println ("Ciao Mondo!");
    }
}
```

supponendo questa volta che sia contenuto nel file '`ciao_mondo.java`', si può generare il binario Java '`CiaoMondoApp.class`' con il comando seguente:

`$` **`gcj -C ciao_mondo.java`** [*Invio*]

Per compilare il binario Java in modo da ottenere un binario adatto al sistema operativo e all'architettura del proprio elaboratore, si può usare il comando seguente, generando quindi l'eseguibile '**`ciao`**':

`$` **`gcj --main=CiaoMondoApp -o ciao CiaoMondoApp.class`** [*Invio*]

Infine, per compilare direttamente il sorgente Java, si può agire nello stesso modo:

`$` **`gcj --main=CiaoMondoApp -o ciao ciao_mondo.java`** [*Invio*]

GCJ riconosce la variabile di ambiente '**`CLASSPATH`**', per la ricerca delle classi, fornendo anche la possibilità di indicare tale informazione attraverso la riga di comando, con delle opzioni che qui non vengono mostrate.

Opzione	Descrizione
`-C`	In questo caso, i file in ingresso sono sorgenti Java e vengono compilati generando le classi in forma di binari Java.
`--main=`***classe***	Questa opzione permette di stabilire quale sia la classe da utilizzare come principale, in modo che il programma che si genera inizi da lì il suo funzionamento.
`-o` ***file***	Definisce il nome dell'eseguibile da generare, quando la compilazione non è destinata a ottenere soltanto un binario Java.

Riferimenti

«

- *TransVirtual Technologies Inc.*
  http://www.transvirtual.com
- Riferimenti per ottenere il JDK dalla rete
  http://www.blackdown.org/
- *The source for Java, Documentation*
  http://java.sun.com/docs/index.html
- *The source for Java, Tutorial*
  http://java.sun.com/docs/books/tutorial/index.html

[1] **Kaffe** software libero con licenza speciale

[2] **JDK** software non libero

[3] **GCJ** GNU GPL