Java networking

Si sente spesso affermare che Java è “il linguaggio di programmazione per Internet”. Effettivamente la maggior parte del grande successo e della diffusione di Java è dovuta a questo, vista soprattutto l’importanza sempre maggiore che Internet sta assumendo. Java è quindi particolarmente adatto per sviluppare applicazioni che devono fare uso della rete. Ciò non deve indurre a pensare che con Java si scrivono principalmente solo Applet, per animare e rendere più carine e interattive le pagine web. Con Java si possono sviluppare vere e proprie applicazioni che devono girare in rete interagendo con più computer (le cosiddette applicazioni distribuite).

Non si dimentichi che un altro fattore determinante per il suo successo è l’indipendenza dalla piattaforma, ottenuta grazie all’utilizzo del bytecode. Il linguaggio astrae da problemi di portabilità come il byte ordering, e quindi anche il programmatore non deve preoccuparsi dei classici problemi di interoperabilità cross-platform.

A questo punto il programmatore di una applicazione network based non deve preoccuparsi di scrivere ex novo particolari librerie o funzioni per le operazioni di base, ma può dedicarsi totalmente ai dettegli veri e propri dell’applicazione.

Inoltre ciò che rende Java un linguaggio adatto per il networking sono le classi definite nel pacchetto java.net che sarà analizzato in questo capitolo, in cui, oltre alla descrizione delle varie classi e dei rispettivi metodi, saranno forniti anche semplici esempi estendibili e funzionanti.

Socket

Le classi di networking incapsulano il paradigma socket presentato per la prima volta nella Berkeley Software Distribution (BSD) della University of California at Berkeley. Una socket è come una porta di comunicazione e non è molto diversa da una presa elettrica: tutto ciò che è in grado di comunicare tramite il protocollo standard TCP/IP può collegarsi ad una socket e comunicare tramite questa porta, allo stesso modo in cui un qualsiasi apparecchio che funziona a corrente può collegarsi a una presa elettrica e sfruttare la tensione messa a disposizione. Nella “rete” gestita dalle socket, invece dell’elettricità, viaggiano pacchetti TCP/IP. Tale protocollo e le socket forniscono quindi un’astrazione

che permette di far comunicare dispositivi diversi che utilizzano lo stesso protocollo.
Quando si parla di networking, ci si imbatte spesso nel termine client-server. Si tratta in realtà di un paradigma: un'entità (spesso un programma) client per portare a termine un particolare compito richiede dei servizi ad un'altra entità (anche questa spesso è un programma): un server che ha a disposizione delle risorse da condividere. Una tale situazione si ritrova spesso nell'utilizzo quotidiano dei computer (anche senza saperlo): un programma che vuole stampare qualcosa (client) richiede alla stampante (server) l'utilizzo di tale risorsa.
Il server è una risorsa costantemente disponibile, mentre il client è libero di scollegarsi dopo che è stato servito. Tramite le socket inoltre un server è in grado di servire più client contemporaneamente.
Alcuni esempi di client-server molto noti sono:
Telnet : se sulla nostra macchina si ha disposizione il programma Telnet (programma client), è possibile operare su un computer remoto come si opera su un computer locale. Questo è possibile se sulla macchina remota è presente un programma server in grado di esaudire le richieste del client Telnet;

FTP : tramite un client FTP si possono copiare e cancellare files su un computer remoto, purché qui sia presente un server FTP;
Web : il browser è un client web, che richiede pagine web ai vari computer su cui è installato un web server, che esaudirà le richieste spedendo la pagina desiderata.

Il client deve conoscere l'indirizzo del server
e il particolare protocollo di comunicazione utilizzato dal server. L'indirizzo in questione è un classico indirizzo IP.
Un client, quindi, per comunicare con un server usando il protocollo TCP/IP dovrà per prima cosa creare una socket con tale server, specificando l'indirizzo IP della macchina su cui il server è in esecuzione e il numero di porta sulla quale il server è in ascolto. Il concetto di porta permette ad un singolo computer di servire più client contemporaneamente: su uno stesso computer possono essere in esecuzione server diversi, in ascolto su porte diverse. Se si vuole un'analogia si può pensare al fatto che più persone abitano

nella medesima via, ma a numeri civici diversi. In questo caso i numeri civici rappresenterebbero le porte.

Un server “rimarrà in ascolto” su una determinata porta finché un client non creerà una socket con la macchina del server, specificando quella porta. Una volta che il collegamento con il server, tramite la socket è avvenuto, il client può iniziare a comunicare con il server, sfruttando la socket creata. A collegamento avvenuto si instaura un protocollo di livello superiore che dipende da quel particolare server: il client deve utilizzare quel protocollo di comunicazione, per richiedere servizi al server.

Il numero di porta è un intero compreso fra 1 e 65535. Il TCP/IP riserva le porte minori di 1024 a servizi standard. Ad esempio la porta 21 è riservata all’FTP, la 23 al Telnet, la 25 alla posta elettronica, la 80 all’HTTP (il protocollo delle pagine web), la 119 ai news server. Si deve tenere a mente che una porta in questo contesto non ha niente a che vedere con le porte di una macchina (porte seriali, parallele, ecc.), ma è un’astrazione utile per smistare informazioni a più server in esecuzione su una stessa macchina.

Si presentano adesso le classi messe a disposizione da Java nel pacchetto java.net per la gestione di comunicazioni in rete.

La classe InetAddress

Come si sa, un indirizzo Internet è costituito da 4 numeri (da 0 a 255) separati ciascuno da un punto. Spesso però, quando si deve accedere a un particolare host, invece di specificare dei numeri, si utilizza un nome, che corrisponde a tale indirizzo (p.e.: www.myhost.it). La traduzione dal nome all’indirizzo numerico vero e proprio è compito del servizio Domain Name Service, abbreviato con DNS.

Senza entrare nei dettagli di questo servizio, basti sapere che la classe InetAddress mette a disposizione diversi metodi per astrarre dal particolare tipo di indirizzo specificato (a numeri o a lettere), occupandosi essa stessa di effettuare le dovute traduzioni. Inoltre c’è un ulteriore vantaggio: la scelta di utilizzare un indirizzo numerico a 32 bit non fu a suo tempo una scelta molto lungimirante; con l’immensa diffusione che Internet ha avuto e sta avendo, si è molto vicini ad esaurire tutti i possibili indirizzi che si possono ottenere con 32 bit (oltretutto diversi indirizzi sono riservati e quindi il numero di indirizzi possibili si riduce ulteriormente); si stanno pertanto introducendo degli indirizzi a

128 bit che, da questo punto di vista, non dovrebbero più dare tali preoccupazioni.
Le applicazioni che utilizzeranno indirizzi Internet tramite la classe InetAddress saranno portabili dal punto di vista degli indirizzi, in modo completamente trasparente.
Descrizione classe

public final class InetAddress extends Object implements Serializable

Costruttori

La classe non mette a disposizione nessun costruttore: l'unico modo per creare un oggetto InetAddress prevede l'utilizzo di metodi statici, descritti di seguito.

Metodi

public static InetAddress getByName(String host) throws UnknownHostException

Restituisce un oggetto InetAddress rappresentante l'host specificato nel parametro host . L'host può essere specificato sia come nome, che come indirizzo numerico. Se si specifica null come parametro, ci si riferisce all'indirizzo di default della macchina locale.

public static InetAddress[] getAllByName(String host) throws UnknownHostException

Tale metodo è simile al precedente, ma restituisce un array di oggetti InetAddress : spesso alcuni siti web molto trafficati registrano lo stesso nome con indirizzi IP diversi. Con questo metodo si otterranno tanti InetAddress quanti sono questi indirizzi registrati.

public static InetAddress getLocalHost() throws UnknownHostException

Viene restituito un InetAddress corrispondente alla macchina locale. Se tale macchina non è registrata, oppure è protetta da un firewall, l'indirizzo è quello di loopback: 127.0.0.1 .

Tutti questi metodi possono sollevare l'eccezione UnknownHostException se l'indirizzo specificato non può essere risolto (tramite il DNS).

public String getHostName()

Restituisce il nome dell'host che corrisponde all'indirizzo IP dell' InetAddress .

Se il nome non è ancora noto (ad esempio se l'oggetto è stato creato specificando un indirizzo IP numerico), verrà cercato tramite il DNS; se tale ricerca fallisce, verrà restituito l'indirizzo IP numerico (sempre sotto forma di stringa).

public String getHostAddress()

Simile al precedente: restituisce però l’indirizzo IP numerico, sotto forma di stringa, corrispondente all’oggetto InetAddress .

public byte[] getAddress()

L’indirizzo IP numerico restituito sarà sotto forma di array byte. L’ordinamento dei byte è high byte first (che è proprio l’ordinamento tipico della rete).

Un’Applet potrà costruire un oggetto InetAddress solo per l’host dove si trova il web server dal quale l’Applet è stata scaricata, altrimenti verrà generata un’eccezione: SecurityException .

Un esempio

Con tale classe a disposizione è molto semplice scrivere un programma in grado di tradurre nomi di host nei corrispettivi indirizzi numerici e viceversa. Al programma che segue basterà passare una stringa contenente o un nome di host o un indirizzo IP numerico e si avranno in risposta le varie informazioni.

```
import java.net.*;
import java.io.*;
public class HostLookup {
public static void main(String args[]) {
// prima si stampano i dati relativi
// alla macchina locale...
try {
InetAddress LocalAddress = InetAddress.getLocalHost();
System.out.println("host locale : "
+ LocalAddress.getHostName() + ", IP : "
+ LocalAddress.getHostAddress());
} catch(UnknownHostException e) {
System.err.println("host locale sconosciuto!");
e.printStackTrace();
}
// ...poi quelli dell’host specificato
```

```
if(args.length != 1) {
System.err.println("Uso: HostLookup host");
} else {
try {
System.out.println("Ricerca di " + args[0] + "...");
InetAddress RemoteMachine = InetAddress.getByName(args[0]);
System.out.println("Host Remoto : "
+ RemoteMachine.getHostName() + ", IP : "
+ RemoteMachine.getHostAddress() );
} catch(UnknownHostException ex) {
System.out.println("Ricerca Fallita " + args[0]);  }  }  }  }
```

URL

Tramite un URL (Uniform Resource Locator) è possibile riferirsi alle risorse di Internet in modo semplice e uniforme. Si ha così a disposizione una forma intelligente e pratica per identificare o indirizzare in modo univoco le informazioni su Internet. I browser utilizzano gli URL per recuperare le pagine web. Java mette a disposizione alcune classi per utilizzare gli URL; sarà così possibile, ad esempio, inglobare nelle proprie applicazioni funzioni tipiche dei web browser.

Esempi tipici di URL sono:

http://www.myweb.com:8080/webdir/webfile.html

ftp://ftp.myftpsite.edu/pub/programming/tips.tgz

Un URL consiste di 4 componenti:

1. il protocollo separato dal resto dai due punti (esempi tipici di protocolli sono http , ftp , news , file , ecc.);
2. il nome dell’host, o l’indirizzo IP dell’host, che è delimitato sulla sinistra da due barre (//), e sulla destra da una sola (/), oppure da due punti (:);
3. il numero di porta, separato dal nome dell’host sulla sinistra dai due punti, e sulla destra da una singola barra. Tale componente è opzionale, in quanto, come già detto, ogni protocollo ha una porta di default;
4. il percorso effettivo della risorsa che richiediamo. Il percorso viene specificato come si specifica un path sotto Unix. Se non viene specificato nessun file, la

maggior parte dei server HTTP aggiunge automaticamente come file di default

Descrizione classe

public final class URL extends Object implements Serializable

Costruttori

La classe ha molti costruttori, poiché vengono considerati vari modi di specificare un URL.

public URL(String spec) throws MalformedURLException

L'URL viene specificato tramite una stringa, come ad esempio:

http://www.myweb.it:80/foo.html

public URL(URL context, String spec) throws MalformedURLException

L'URL viene creato combinando un URL già esistente e un URL specificato tramite una stringa. Se la stringa è effettivamente un URL assoluto, allora l'URL creato corrisponderà a tale stringa; altrimenti l'URL risultante sarà il percorso specificato in spec , relativo all'URL context . Ad esempio se context è http://www.myserver.it/ e spec è path/index.html , l'URL risultante sarà http://www.myserver.it/path/ index.html .

public URL(String protocol, String host, int port,String file) throws MalformedURLException

Con questo costruttore si ha la possibilità di specificare separatamente ogni singolo componente di un URL.

public URL(String protocol, String host, String file) throws MalformedURLException

Simile al precedente, ma viene usata la porta di default del protocollo specificato.

Un'eccezione MalformedURLException viene lanciata se l'URL non è specificato in modo corretto (per quanto riguarda i vari componenti).

Metodi

Di questa classe fanno parte diversi metodi che permettono di ricavare le varie parti di un URL

public int getPort()

public String getProtocol()

public String getHost()

public String getFile()

Il significato di questi metodi dovrebbe essere chiaro: restituiscono un singolo componente dell'oggetto URL.

public String toExternalForm()

Restituisce una stringa che rappresenta l'URL

public URLConnection openConnection() throws IOException

Restituisce un oggetto URLConnection (sarà trattato di seguito), che rappresenta una connessione con l'host dell'URL, secondo il protocollo adeguato. Tramite questo oggetto, si può accedere ai contenuti dell'URL.

public final InputStream openStream() throws IOException

Apre una connessione con l'URL, e restituisce un input stream. Tale stream può essere utilizzato per leggere i contenuti dell'URL.

public final Object getContent() throws IOException

Questo metodo restituisce un oggetto di classe Object che racchiude i contenuti dell'URL. Il tipo reale dell'oggetto restituito dipende dai contenuti dell'URL: se si tratta di un'immagine, sarà un oggetto di tipo Image , se si tratta di un file di testo, sarà una String . Questo metodo compie diverse azioni, invisibili all'utente, come stabilire la connessione con il server, inviare una richiesta, processare la risposta, ecc.

Un esempio

Ovviamente per ogni protocollo ci dovrà essere un appropriato gestore. Il JDK fornisce di default un gestore del protocollo HTTP, e quindi l'accesso alle informazioni web è alquanto semplice.

Nel caso dell'HTTP, ad esempio una chiamata al metodo openStream , il gestore del protocollo HTTP, invierà una richiesta al web server specificato con l'URL, analizzerà le risposte del server, e restituirà un input stream dal quale è possibile leggere i contenuti del particolare file richiesto. Richiedere un file a un server web è molto semplice, ed è illustrato nell'esempio seguente, che mostra anche l'utilizzo di altri metodi della classe.

```
import java.net.*;
import java.io.*;
public class HTTP_URL_Reader {
public static void main(String args[]) throws IOException {
if(args.length < 1)
throw new IOException("Sintassi : HTTP_URL_Reader URL");
URL url = new URL(args[0]);
```

```
System.out.println("Componenti dell’URL");
System.out.println("URL
: " + url.toExternalForm());
System.out.println("Protocollo: " + url.getProtocol());
System.out.println("Host
: " + url.getHost());
System.out.println("Porta
: " + url.getPort());
System.out.println("File
: " + url.getFile());
System.out.println("Contenuto dell’URL :");
// lettura dei dati dell’URL
InputStream iStream = url.openStream();
DataInputStream diStream = new DataInputStream(iStream);
String line ;
while((line = diStream.readLine()) != null)
System.out.println(line);
diStream.close(); } }
```

È sufficiente creare un URL, passando al costruttore l’URL sotto forma di stringa, ottenere l’input stream chiamando l’apposito metodo, creare un DataInputStream basandosi su tale stream, e leggere una riga alla volta, stampandola sullo schermo. Il programma può essere eseguito così:

java HTTP_URL_Reader http://localhost/mydir/myfile.html

Se è installato un web server, si avranno stampate a schermo le varie componenti dell’URL specificato, nonché il contenuto del file richiesto.

La classe URLConnection

Questa classe rappresenta una connessione attiva, specifica di un dato protocollo, a un oggetto rappresentato da un URL. Tale classe è astratta, e quindi, per gestire uno specifico protocollo, si dovrebbe derivare da questa classe.

Descrizione classe

public class URLConnection extends Object

Costruttori

protected URLConnection(URL url)

Crea un oggetto di questa classe, dato un URL. Da notare che il costruttore è protetto, quindi può essere chiamato solo da una classe derivata. In effetti, come si è visto nella classe URL, un oggetto di questa classe viene ottenuto tramite la chiamata del metodo openConnection della classe URL .

Metodi

public URL getURL()

Restituisce semplicemente l'URL su cui è stato costruito l'oggetto URLConnection .

public abstract void connect() throws IOException

Permette di connettersi all'URL, specificato nel costruttore. La connessione quindi non avviene con la creazione dell'oggetto, ma avviene quando viene richiamato questo metodo, oppure un metodo che necessita che la connessione sia attiva (a quel punto la richiesta della connessione viene stabilita implicitamente).

public Object getContent() throws IOException

Restituisce il contenuto dell'URL. Viene restituito un oggetto di classe Object , poiché il tipo dell'oggetto dipende dal particolare URL.

public InputStream getInputStream() throws IOException

Restituisce un input stream con il quale si può leggere dall'URL.

public OutputStream getOutputStream() throws IOException

In questo caso si tratta di uno stream di output, con il quale è possibile inviare dati a un URL; ciò può risultare utile se si deve compiere un'operazione di post HTTP.

Questa classe contiene inoltre molti metodi che permettono di avere informazioni dettagliate sull'URL, quali il tipo di contenuto, la sua lunghezza, la data dell'ultima modifica, ecc. Per una rassegna completa si rimanda ovviamente alla documentazione on-line ufficiale.