

Applicazioni distribuite

Maurizio Cozzetto

1 agosto 2009

Un pò di teoria

Ricordiamo che un'applicazione distribuita è un'applicazione composta da più programmi (almeno 2) posti in esecuzione su macchine diverse all'interno di una rete di calcolatori e che utilizzano un protocollo di comunicazione comune. I pc connessi a Internet comunicano tra loro usando diversi protocolli tra cui il più famoso è forse il *TCP/IP* (*Transmission Control Protocol/Internet Protocol*) ben supportato da Java. Un altro protocollo è l'*UDP* (*User Datagram Protocol*).

Il protocollo *TCP/IP* viene definito protocollo *connection-oriented*, orientato alla connessione, cioè garantisce che i pacchetti inviati arrivino a destinazione (se vi è un qualche errore il destinatario chiede la ritrasmissione del pacchetto), mentre l'*UDP* non garantisce l'arrivo di tutti i pacchetti e viene definito *connectionless*. E' preferito in genere in tutti quei casi in cui perdite contenute di informazione durante la trasmissione sono trascurabili (come nelle trasmissioni *audio-video*) e quello che conta invece è la *velocità* di trasmissione.

Java utilizza per la comunicazione in rete il paradigma di programmazione basato sul concetto di *socket*, progettato presso l'*Università di Berkley* (California).

Socket

Il *socket* può essere definito come un *punto terminale* o di *aggancio* di una linea di comunicazione da cui un programma può inviare i dati in rete e può ricevere i dati dalla rete.

Il *socket* (la traduzione è *presa di corrente*) è un'astrazione software che è in qualche modo l'equivalente della presa elettrica per gli apparecchi elettrici.

In pratica un *socket* identifica un *computer* e un *processo* sul computer; è formato da un *indirizzo IP* che identifica un *computer* sulla rete e da un *numero di porta TCP* che identifica un *processo* o un'*applicazione* sul computer.

Il *socket* rappresenta solo uno dei due capi di una comunicazione; due computer per comunicare usano ciascuno un *socket* e tra i due *socket* si crea una *connessione* per il trasferimento dei dati in due direzioni.

La realizzazione di applicazioni client/server in Java si basa sulle classi contenute nel package *java.net* e in particolare sulle classi *InetAddress*, *ServerSocket* e *Socket*.

Ricordiamo brevemente che un programma è detto *client* se richiede un servizio a un programma detto *server* che fornisce il servizio richiesto e poi si rimette *in ascolto* in attesa di ulteriori richieste da parte di altri client. Può accadere frequentemente che su un server vi siano in esecuzione servizi diversi per cui, per distinguerli, si usa un numero identificativo del servizio, la *porta* appunto.

Per esempio su una macchina server, potrebbero *girare* contemporaneamente il server web *Apache*, il server web *Tomcat* (un server web basato su Java) e il database server *MySQL*. Le porte impiegate sono la porta *80* per il server web, la porta *8080* (ma potrebbe essere diversa) per il server Tomcat e la porta *3306* per il database server MySQL. Le porte dalla 0 alla 1023 sono riservate ai principali servizi di rete conosciuti e sono dette anche *well known ports*.

Inoltre un server *singlethread* (o server *unicast*) è un server che comunica con un solo client alla volta (se esistono altri client questi devono attendere il loro turno), mentre se il server è *multithread* (o server *multicast*) al server possono

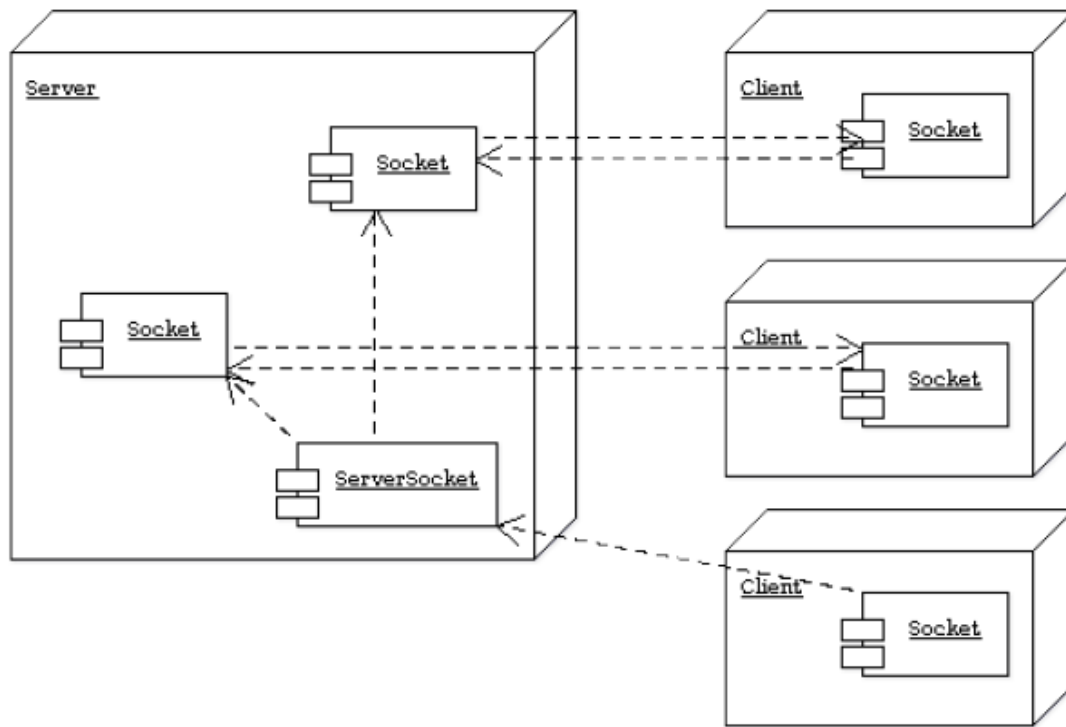


Figura 1: Diagramma UML del meccanismo di comunicazione

connettersi *contemporaneamente* più client (per ogni client, il server attiva un thread *distinto*). In quest'ultimo caso il server sposta la richiesta dalla porta sulla quale è stata effettuata la connessione su una nuova porta.

Altra considerazione importante è che, una volta stabilita la connessione, la comunicazione tra client e server è basata sulla comunicazione tra flussi (*stream*), cioè l'invio e la ricezione dei dati sono gestiti in maniera simile alla gestione dei file.

L'applicazione

L'applicazione che presentiamo consiste di 3 thread: *ClientThreadMax*, che rappresenta il client, *ServerThreadMax*, che rappresenta il server e *ThreadMax* che rappresenta il thread del servizio richiesto dal client (in questo caso, il servizio richiesto è banalmente il massimo tra due numeri interi) e ingloba la *logica* dell'applicativo.

Il funzionamento è il seguente: *ServerThreadMax* consiste di un ciclo *infinito* di attesa di una qualche connessione sulla porta 10000 (un valore scelto a caso, qualsiasi altro valore maggiore di 1024 sarebbe andato altrettanto bene) mediante un oggetto di classe *ServerSocket* (*righe dalla 22 alla 36*). Una volta stabilita la connessione col client, rappresentata da un oggetto di classe *Socket* (*riga 26*), istanzia un thread di servizio per quel client e lo fa partire (*righe 34 e 35*). Poi torna a mettersi in ascolto sulla porta 10000.

Il client *ClientThreadMax* crea una connessione col server all'indirizzo IP e sulla porta indicati nell'oggetto di classe *Socket* (*riga 18*). Poi genera due valori interi casuali e li invia, mediante lo stream di output associato al socket, al server (*righe dalla 29 alla 35*). Il server provvede ad elaborare i dati e li rinvia al client che li riceve sullo stream di input (*riga 42*).

Il thread *ThreadMax* provvede a ricevere sullo stream di input associato i due valori (*righe 26 e 27*), ne calcola il risultato e provvede a inviarlo al client attraverso lo stream di output (*righe dalla 30 alla 33*).

Tutto questo in maniera *concorrente* con altri thread (*parallelismo virtuale* se vi è una sola CPU, *reale* se vi sono più CPU e la JVM supporta il *multithreading reale*).

Il codice

L'applicativo è stato testato in locale usando NetBeans 6.7, un ottimo IDE per lo sviluppo di applicazioni in Java (e non solo) e comprende, come già detto, le classi *ServerThreadMax*, *ClientThreadMax* e la classe *ThreadMax*. Di ogni classe per brevità riportiamo i frammenti essenziali (il progetto completo *ClientServerMultiThreadJavaApp6* è scaricabile dal sito o direttamente dal link precedente).

Classe ServerThreadMax

```
1  /*
2  * ServerThreadMax.java
3  */
4  package it.maurziocozzetto.classi;
5  import java.io.IOException;
6  import java.net.ServerSocket;
7  import java.net.Socket;
8
9  public class ServerThreadMax extends Thread {
10     final static int port = 10000;
11     ServerSocket serverSocket;
12
13     public ServerThreadMax() throws IOException {
14         // istanza di un oggetto di classe ServerSocket
15         serverSocket = new ServerSocket(port);
16     }
17
18     @Override
19     public void run() {
20         try {
21             // ciclo infinito di attesa delle connessioni dei client
22             while (true) {
23
24                 // il server si sospende e rimane in
25                 // attesa della connessione
26                 Socket socket = serverSocket.accept();
27
28                 // ottengo alcune utili informazioni sulla connessione
29                 String addr = socket.getInetAddress().toString();
30                 int remotePort = socket.getPort();
31                 int localPort = socket.getLocalPort();
32
33                 // istanzio il thread di servizio e lo faccio partire
34                 ThreadMax threadMax = new ThreadMax(socket);
35                 threadMax.start();
36             } // fine ciclo infinito
37         } catch (IOException e) {
38             // Errore di IO
39         } // fine try-catch
40     } // fine metodo run
41
42 } // fine classe ServerThreadMax
```

Classe ClientThreadMax

```
1  /*
2   * ClientThreadMax.java
3   */
4  package it.mauriziocozzetto.classi;
5  import java.io.PrintStream;
6  import java.net.Socket;
7  import java.util.Scanner;
8
9  public class ClientThreadMax extends Thread {
10     static final int port = 10000;
11     Socket socket;
12     Scanner in;
13     PrintStream out;
14     String threadName;
15
16     public ClientThreadMax(String threadName) throws UnknownHostException,
17     IOException {
18         this.threadName=threadName;
19         // creo il socket
20         socket = new Socket("localhost", port);
21
22         // preparo gli stream di input e di output del client
23         in = new Scanner(socket.getInputStream());
24         out = new PrintStream(socket.getOutputStream());
25     } // fine costruttore
26
27     @Override
28     public void run() {
29         try{
30             // informazioni sulla connessione
31             String addr = socket.getInetAddress().toString();
32             int remotePort = socket.getPort();
33             int localPort = socket.getLocalPort();
34             // genero due numeri interi casuali compresi tra 0 e 99
35             int n1 = (int)(Math.random()*(99));
36             int n2 = (int)(Math.random()*(99));
37             // il client invia al server i due valori
38             // usando il canale di output
39             out.println(n1);
40             out.println(n2);
41             // il server elabora il calcolo (in questo caso il massimo)
42             // creando un thread apposito (un oggetto di tipo ThreadMax)
43             // il client riceve il risultato del calcolo sul
44             // canale di input
45             String maxStr = in.nextLine();
46             // chiudo il socket e gli stream di IO
47             socket.close();
48             in.close();
49             out.close();
50         } catch(IOException e){
51             // Eccezione di IO
52         } // fine blocco try-catch
53     } // fine metodo run
54 } // fine classe ClientThreadMax
```

Classe ThreadMax

```
1  /*
2   * ThreadMax.java
3   */
4  package it.mauriziocozzetto.classi;
5  import java.io.IOException;
6  import java.io.PrintStream;
7  import java.net.Socket;
8  import java.util.Scanner;
9
10 public class ThreadMax extends Thread {
11     Socket socket;
12     Scanner in;
13     PrintStream out;
14
15     public ThreadMax(Socket socket) throws IOException {
16         this.socket = socket;
17         // istanzio i due stream di IO
18         in = new Scanner(this.socket.getInputStream());
19         out = new PrintStream(this.socket.getOutputStream());
20     } // fine costruttore
21
22     @Override
23     public void run() {
24         try {
25             // informazioni utili sulla connessione
26             String addr = socket.getInetAddress().toString();
27             int remotePort = socket.getPort();
28             int localPort = socket.getLocalPort();
29
30             // leggo il primo e il secondo numero
31             int n1= Integer.parseInt(in.nextLine());
32             int n2= Integer.parseInt(in.nextLine());
33
34             // qui si concentra tutta la logica del servizio
35             int max = Math.max(n1, n2);
36
37             // mando in output sullo stream il risultato
38             out.println(max);
39
40             // chiudo la connessione col socket e gli stream di IO
41             socket.close();
42             in.close();
43             out.close();
44         } catch (IOException e){
45             // Eccezione di IO
46         } // fine blocco try-catch
47     } // fine metodo run
48
49 }
```

Esempio di esecuzione con 2 thread

```
ServerThreadMax ==> Tento di creare l'oggetto di classe ServerThreadMax
ServerThreadMax ==> Oggetto di classe ServerThreadMax creato
ClientThreadMax-1 ==> Tento di creare il socket per connettermi al server...
ServerThreadMax ==> ServerThreadMax attivo e in ascolto sulla porta 10000...
ServerThreadMax ==> Indirizzo del client: /127.0.0.1
ServerThreadMax ==> Porta del client: 59322
ServerThreadMax ==> Porta locale: 10000
ClientThreadMax-1 ==> Socket creato
ServerThreadMax ==> Creato nuovo ThreadMax per servire il client
ServerThreadMax ==> ThreadMax avviato per il nuovo client
ServerThreadMax ==> ServerThreadMax attivo e in ascolto sulla porta 10000...
ThreadMax-1 ==> Indirizzo del client: /127.0.0.1
ThreadMax-1 ==> Porta remota: 59322
ThreadMax-1 ==> Porta locale: 10000
ThreadMax-1 ==> ThreadMax partito. Calcolo del massimo tra due numeri
ClientThreadMax-2 ==> Tento di creare il socket per connettermi al server...
ServerThreadMax ==> Indirizzo del client: /127.0.0.1
ServerThreadMax ==> Porta del client: 59323
ServerThreadMax ==> Porta locale: 10000
ServerThreadMax ==> Creato nuovo ThreadMax per servire il client
ServerThreadMax ==> ThreadMax avviato per il nuovo client
ServerThreadMax ==> ServerThreadMax attivo e in ascolto sulla porta 10000...
ThreadMax-2 ==> Indirizzo del client: /127.0.0.1
ThreadMax-2 ==> Porta remota: 59323
ThreadMax-2 ==> Porta locale: 10000
ThreadMax-2 ==> ThreadMax partito. Calcolo del massimo tra due numeri
ClientThreadMax-1 ==> Indirizzo del server: localhost/127.0.0.1
ClientThreadMax-1 ==> Porta del server: 10000
ClientThreadMax-1 ==> Porta locale: 59322
ClientThreadMax-1 ==> Genero n1
ClientThreadMax-1 ==> n1=49
ClientThreadMax-1 ==> Genero n2
ClientThreadMax-1 ==> n2=49
ClientThreadMax-1 ==> Invio i due valori al server
ClientThreadMax-1 ==> Ricevo dal server il risultato
ThreadMax-1 ==> Ho letto n1=49 dal client
ThreadMax-1 ==> Ho letto n2=49 dal client
ThreadMax-1 ==> Calcolo il max
ThreadMax-1 ==> Il max vale 49
ThreadMax-1 ==> Invio il max al client
ThreadMax-1 ==> Max inviato
ThreadMax-1 ==> Chiudo la connessione col client e chiudo gli stream di IO
ClientThreadMax-1 ==> Il massimo è: 49
ClientThreadMax-2 ==> Socket creato
```

ClientThreadMax-2 ==> Indirizzo del server: localhost/127.0.0.1
ClientThreadMax-2 ==> Porta del server: 10000
ClientThreadMax-2 ==> Porta locale: 59323
ClientThreadMax-2 ==> Genero n1
ClientThreadMax-2 ==> n1=90
ClientThreadMax-2 ==> Genero n2
ClientThreadMax-2 ==> n2=64
ClientThreadMax-2 ==> Invio i due valori al server
ClientThreadMax-2 ==> Ricevo dal server il risultato
ThreadMax-2 ==> Ho letto n1=90 dal client
ThreadMax-2 ==> Ho letto n2=64 dal client
ThreadMax-2 ==> Calcolo il max
ThreadMax-2 ==> Il max vale 90
ThreadMax-2 ==> Invio il max al client
ThreadMax-2 ==> Max inviato
ThreadMax-2 ==> Chiudo la connessione col client e chiudo gli stream di IO
ClientThreadMax-2 ==> Il massimo è: 90