Programmazione in rete

Introduzione…

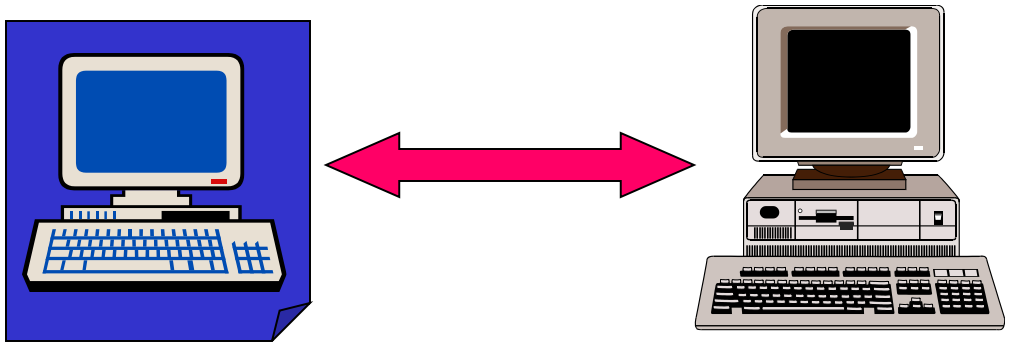
Storicamente, la programmazione in rete (o programmazione distribuita) è stata sempre complessa e soggetta ad errori. Le principali difficoltà sono dovute alla necessità, da parte del programmatore, di conoscere i dettagli della rete, dei protocolli e persino dell'hardware. Al programmatore si richiede di utilizzare delle librerie di funzioni per connettersi a un nodo della rete, per impacchettare e spacchettare i messaggi, per inviare i messaggi, tutto secondo rigidi protocolli di

handshaking.

Introduzione…

In Java la programmazione in rete è notevolmente semplificata e astratta molto bene in un insieme di classi.

I progettisti di Java hanno reso la programmazione in rete molto simile alla lettura e scrittura di file, con la differenza che i “file” esistono su un elaboratore remoto e che questo può decidere esattamente cosa vuole fare dell’informazione richiesta o inviata.



Introduzione…

Il modello di programmazione usato è quello di un file; infatti si fa il wrapping di una connessione di rete (un ***socket***) in un flusso (stream) di oggetti, in modo da utilizzare le stesse invocazioni di metodo utilizzate per i flussi di oggetti al fine di scambiare informazioni.

Grazie al fatto che Java è multipiattaforma, i dettagli relativi alla rete sono stati astratti e "presi in carico" dalla JVM e dalla installazione locale di Java.

Infine le caratteristiche multithreading di Java facilitano un altro aspetto importante della programmazione in rete: la gestione di connessioni multiple concorrenti.

Ma procediamo con ordine …

Identificazione di una macchina…

Sicuramente, al fine di comunicare con un altro nodo della rete, è necessario connettersi con l'elaboratore giusto. Occorre dunque essere in grado di identificarlo univocamente.

L'identificazione del nodo avviene mediante l'IP (Internet Protocol).

Esistono due modi per identificare l'IP:

- Mediante DNS (*Domain Name System*). Per es. *appice.di.uniba.it*
- Mediante *dot notation*. Ad es. 194.207.187.85

…Identificazione di una macchina.

In Java si usa una speciale classe per rappresentare l'IP in entrambe le forme: **InetAddress** del package **java.net**.

La classe dispone di un metodo statico **InetAddress.getByName()** che permette di ottenere un oggetto **InetAddress** a partire al nome o dall'indirizzo IP di un host.

Esempio

```
InetAddress addr= InetAddress.getByName(null);
```

restituisce l'indirizzo IP del "localhost"

```
InetAddress addr=
  InetAddress.getByName("127.0.0.1");
```

Uso del Port

Un indirizzo IP non è sufficiente per individuare un server unico. Infatti possono esistere più server su una stessa macchina.

Quando si imposta un client o un server è necessario scegliere la "porta" (*port*) sul quale sia il server che il client decidono di connettersi.

Il *port* non è una locazione fisica su una macchina ma è una astrazione software.

Tipicamente ogni servizio è associato ad un singolo numero di *port* su una macchina server.

Il programma client non deve conoscere soltanto l'indirizzo IP, ma anche il *port* giusto per il servizio richiesto.

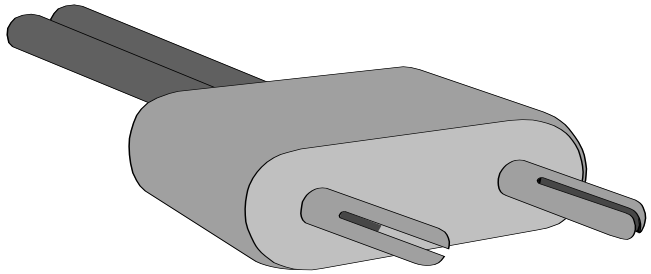
Uso del Port

Esempio:

appice.di.uniba.it:8000 8000 è il port associato ad un MS Personal Web Server

appice.di.uniba.it:8080 8080 è il port associato and un altro server Web

Socket …



In Java si usa un socket per creare la connessione ad un'altra macchina. In particolare, per stabilire una connessione fra due computer occorrerà disporre di un socket su ogni macchina.

Il *socket* è una astrazione software usata per rappresentare i terminali di una connessione tra due macchine.

Creando un socket in Java, si ottengono un **InputStream** e un **OutputStream** (o, con appropriate conversioni, un **Reader** e un **Writer**) al fine di abilitare la connessione in modo simile a un I/O su stream di oggetti.

…Socket …

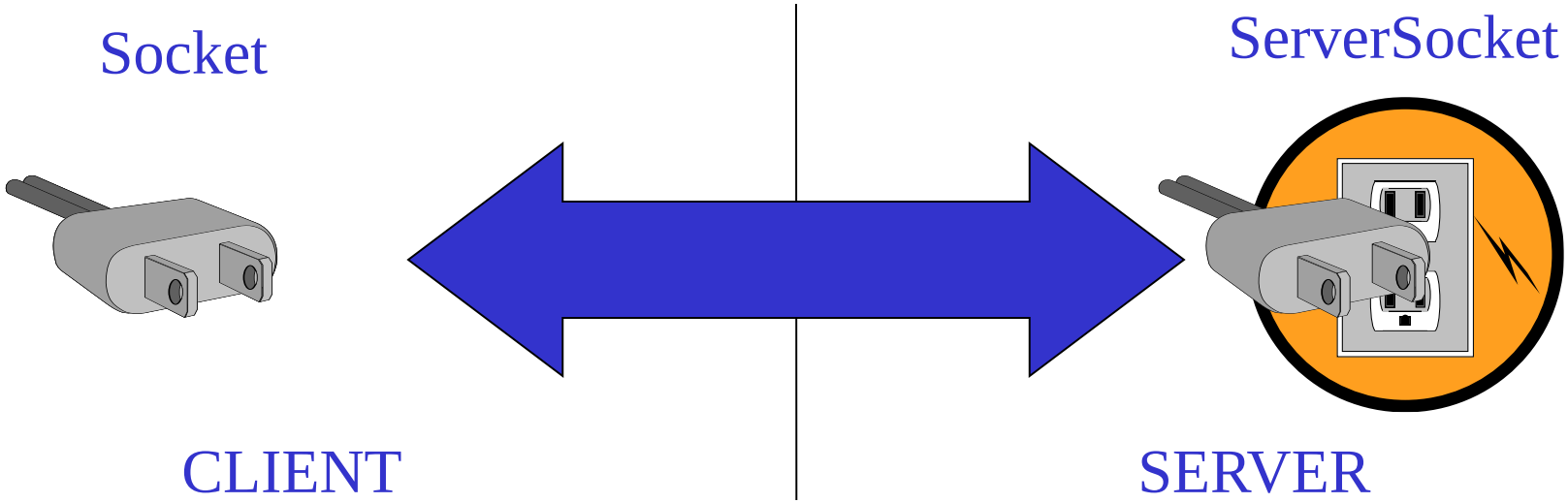
Ci sono due classi socket basate su stream:

- **ServerSocket** che il server usa per ascoltare una richiesta di connessione.
- **Socket** usata dal client per inizializzare la connessione.

Una volta che un client richiede una connessione socket, il **ServerSocket** restituisce (mediante il metodo **accept( )**) un **Socket** corrispondente attraverso il quale la comunicazione può avvenire dal lato server.

Solo dopo che è avvenuto tutto ciò si ha una connessione "**Socket-**to**-Socket**".

…Socket …



Quando si crea un **ServerSocket**, si specifica solo un numero di port. Non occorre specificare un indirizzo IP poiché esso è già associato alla macchina sul quale il server gira. Al contrario, quando si crea un **Socke**t lato client, occorre specificare tanto l'indirizzo IP quanto il numero di port al quale connettersi.

Il socket restituito da **ServerSocket.accept( )** conterrà poi entrambe le informazioni.

…Socket.

A questo punto si usano i metodi **getInputStream()** e **getOutputStream()** per produrre i corrispondenti oggetti delle classi **InputStream** e **OutputStream** a partire dai singoli **Socket**.

Gli stream ottenuti permettono, quindi, di lavorare con classi buffer e classi di formattazione. Proprio come avveniva nell'I/O da file.

Esempio

esempio di server.doc

esempio di client.doc

Servire più client …

Un problema di non poca rilevanza è la necessità di manipolare più connessioni contemporaneamente. Per servire più client contemporaneamente si ricorre al multithreading.

Lo schema base prevede la creazione di un singolo **ServerSocket** sul server e chiamare **accept( )** per attendere una connessione.

Quando la connessione è attiva e **accept( )** termina la sua esecuzione si utilizza il **Socket** ottenuto in un *nuovo thread* utilizzato per servire un particolare client. Il thread principale, intanto, richiamerà **accept( )** per attendere un nuovo client.

Esempio

server con multithreading.doc