JAVA – Programmazione in rete

Metodi Avanzati di Programmazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Docente: Pierpaolo Basile

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. www.di.uniba.it
  - Mediante dot notation. Ad es. 183.201.181.10

…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 dal nome o dall'indirizzo IP di un host

```
//null sta per localhost
InetAddress add = InetAddress.getByName(null);
System.out.println(add);
add = InetAddress.getByName("localhost");
System.out.println(add);
add = InetAddress.getByName("www.google.it");
System.out.println(add);
```

Uso del Port

- Un indirizzo IP non è sufficiente per individuare un server unico: 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

20/tcp	FTP - Il file transfer protocol - data
21/tcp	FTP - Il file transfer protocol - control
22/tcp	SSH - Secure login, file transfer (scp, sftp) e port forwarding
23/tcp	Telnet insecure text communications
25/tcp	SMTP - Simple Mail Transfer Protocol (E-mail)
53/udp	DNS - Domain Name System
67/udp	BOOTP Bootstrap Protocol (Server) e DHCP Dynamic Host Configuration Protocol (Server)
68/udp	BOOTP Bootstrap Protocol (Client) e DHCP Dynamic Host Configuration Protocol (Client)
69/udp	TFTP Trivial File Transfer Protocol
70/tcp	Gopher

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 Socket lato client, occorre specificare tanto l'indirizzo IP quanto il numero di port al quale connettersi
  - Server e client generalmente non risiedono sulla stessa macchina
- 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

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

Esempi

Package di.uniba.map.b.lab.rete

- **EsempioIndirizzo:** esempio di utilizzo della classe InetAddress
- **JabberServer:** esempio di socket server (richiesta singola)
- **MultiJabberServer:** esempio di socket server multi-thread
- **JabberClient:** esempio di socket client


I KNOW
TCP/IP OVER JAVA SOCKET