Socket in Java

Il package Java.net

Il package java.net contiene le classi che implementano le funzionalità di rete, fornisce classi e interfacce per:

- Indirizzamento: `InetAddress`
- Creazione di connessioni TCP: `Socket`, `ServerSocket`
- Creazione di connessioni UDP: `DatagramSocket`
- Localizzazione di risorse tramite URL: `URL`, `URLConnection`
- Sicurezza: `SocketPermission`, `NetPermission`

Classi per la comunicazione TCP

Java fornisce due diverse classi per la comunicazione con il protocollo TCP che rispecchiano la struttura client/ server: `Socket` e `ServerSocket`
La differenza tra le due classi è che `Socket` è utilizzata dal client per stabilire una connessione con il server, mentre `ServerSocket` è utilizzata dal server per ascoltare le richieste di connessione da parte dei client.

ServerSocket

La classe `ServerSocket` è utilizzata dal server per ascoltare le richieste di connessione da parte dei client. Ha quattro costruttori:

- `ServerSocket()`
- `ServerSocket(int port)`
- `ServerSocket(int port, int backlog)`
- `ServerSocket(int port, int backlog, InetAddress bindAddr)`

Il parametro `port` indica la porta su cui il server è in ascolto, mentre `backlog` indica il numero massimo di connessioni in coda che il server può accettare. Il parametro `bindAddr` viene utilizzato da un server multihomed per specificare l'indirizzo IP su cui il server è in ascolto.
I costruttori realizzano tutte le operazioni di `socket()`, `bind()` e `listen()`.

ServerSocket

Il metodo più importante della classe `ServerSocket` è `accept()`, che blocca il thread corrente finché non arriva una richiesta di connessione da parte di un client. Quando arriva una richiesta di connessione, il metodo `accept()` restituisce un oggetto di tipo `Socket` che rappresenta la connessione con il client. Il metodo `accept()` può essere chiamato più volte per accettare più connessioni.

```
ServerSocket server = new ServerSocket(6789);
Socket newconnection = server.accept();
```

Per scrivere e leggere dati sulla connessione è possibile utilizzare gli stream di input e output con `PrintWriter` e `BufferedReader`.

```
PrintWriter out = new PrintWriter(newconnection.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(newconnection.getInputStream()));
```

Socket

La classe `Socket` è utilizzata dal client per stabilire una connessione con il server. Ha due costruttori:

- `Socket(InetAddress address, int port)`
- `Socket(String host, int port)`

Il parametro `address` indica l'indirizzo IP del server, mentre `host` indica il nome del server. Il parametro `port` indica la porta su cui il server è in ascolto.

I costruttori realizzano tutte le operazioni di `socket()` e `connect()`.

Socket

Per scrivere e leggere dati sulla connessione è possibile utilizzare gli stream di input e output con `PrintWriter` e `BufferedReader`.

```
Socket client = new Socket("localhost", 6789);
PrintWriter out = new PrintWriter(client.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
```

Esempio: Server Echo

```
import java.net.*;
import java.io.*;

public class ServerEcho {
    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(6789);
        Socket newconnection = server.accept();
        PrintWriter out = new PrintWriter(newconnection.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(newconnection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            out.println(inputLine);
        }
        out.close();
        in.close();
        newconnection.close();
        server.close();
    }
}
```

Esempio: Client Echo

```
import java.net.*;
import java.io.*;

public class ClientEcho {
    public static void main(String[] args) throws IOException {
        Socket client = new Socket("localhost", 6789);
        PrintWriter out = new PrintWriter(client.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }
        out.close();
        in.close();
        stdIn.close();
        client.close();
    }
}
```

Esercizio: Chat

Adesso che conosciamo le classi per la comunicazione TCP, possiamo implementare una semplice chat tra due utenti. Siccome per avere più utenti in una chat è necessario utilizzare il multithreading, implementeremo **per ora** una chat tra un server e un solo client, come se fossero due utenti che chattano tra di loro.
Voglio che il server chieda in input il nome dell'utente che si connette e che il client chieda in input il nome dell'utente che si connette.
Il server deve inviare un messaggio di benvenuto al client, deve stampare i messaggi ricevuti dal client e deve inviare i messaggi digitati dall'utente al client.
Il client deve stampare i messaggi ricevuti dal server e deve inviare i messaggi digitati dall'utente al server.