

Introduzione ai (Java) socket

Enrico Tassi

slides originali di Gessa, Ghini

Dipartimento di Scienze dell'informazione

April 1, 2009

Mi presento

nome Enrico Tassi

email tassi@cs.unibo.it

web <http://www.cs.unibo.it/~tassi>

titolo di studio dottorato di ricerca in informatica

ricerca Mi occupo di

- ▶ Verifica al calcolatore di correttezza di software e matematica
- ▶ Teoria dei tipi e sua implementazione negli interactive theorem prover
- ▶ Matita (un interactive theorem prover sviluppato a Bologna)

hobby sviluppatore Debian GNU/Linux

<http://qa.debian.org/developer.php?login=gareuselesinge&comaint=yes>

Outline

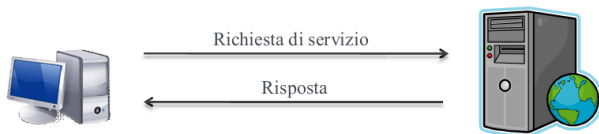
Argomenti

- ▶ reti di calcolatori
- ▶ livelli OSI
- ▶ internet protocol suite
- ▶ socket Java

Scopo della lezione

- ▶ API Java per socket TCP e UDP
- ▶ scrivere un client
- ▶ scrivere un server concorrente

Modello Client/Server



- ▶ Il cosiddetto lato client, effettua la richiesta di un servizio. La sua controparte, il lato server, effettua l'erogazione del servizio richiesto.
- ▶ Chiaramente, è necessario che il lato client e quello server si "intendano" esattamente circa il significato della richiesta e della relativa replica. Si introduce allora il concetto di protocollo.

Architettura dei protocolli ISO/OSI

L'OSI (Open System Interconnections) è un modello per reti emanato dall'ISO (International Standard Organization) alla fine degli anni '70 con lo scopo di essere IL modello di riferimento per le reti di calcolatori.

Cosa vuole definire OSI:

- ▶ la terminologia
- ▶ le funzionalità di una rete
- ▶ base comune per lo sviluppo di standard di rete
- ▶ modello di confronto per le architetture di rete

Architettura dei protocolli ISO/OSI — approccio

Per gestire la complessità dei problemi, l'OSI ha adottato un approccio a livelli (layers).

- ▶ un insieme di 7 livelli
- ▶ ciascun livello esegue funzioni specifiche
- ▶ ciascun livello dialoga con quelli adiacenti



Architettura dei protocolli ISO/OSI — risultati

Cosa ha ottenuto ISO/OSI:

- ▶ ISO ha standardizzato per OSI una serie di protocolli
- ▶ I livelli 1 (Fisico) e 2 (Data Link) sono stati accettati e sono oggi degli standard, garantendo l'interoperabilità di vari prodotti, quali gli switch o le schede di rete.
- ▶ I protocolli di livello superiore (e la loro suddivisione) sono meno accettati.

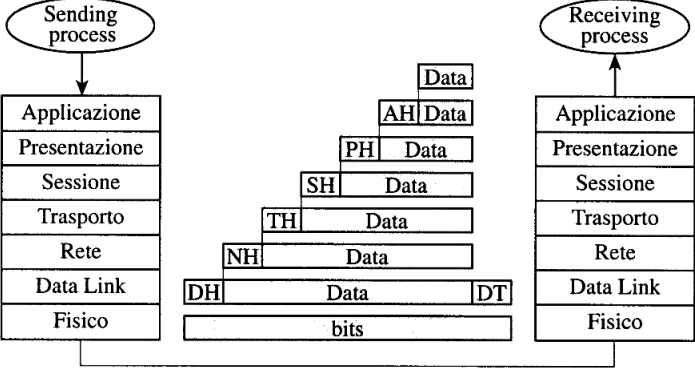
Architettura dei protocolli ISO/OSI — i livelli

- 1-fisico** si occupa di trasmettere sequenze binarie sul canale di comunicazione. Si specificano ad esempio le tensioni che rappresentano 0 e 1 e le caratteristiche dei cavi e dei connettori
- 2-data link** Il livello ha come scopo la trasmissione sufficientemente affidabile di pacchetti (frame) tra due sistemi “contigui”. Accetta come input dei pacchetti di livello 3. Esso verifica la presenza di errori aggiungendo delle checksums e può gestire meccanismi di correzione.
- 3-network** gestisce l'instradamento dei messaggi, ed è il primo livello (a partire dal basso) che gestisce informazioni sulla topologia della rete. Tale livello determina se e quali sistemi intermedi devono essere attraversati dal messaggio per giungere a destinazione (tabelle di instradamento, rotte alternative per fault tolerance)

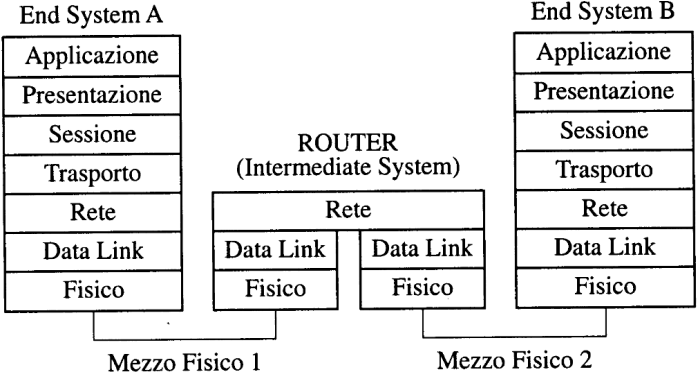
Architettura dei protocolli ISO/OSI — i livelli

- 4-trasporto** servizio di trasferimento trasparente dei dati tra entità del livello 5. Si occupa di garantire un servizio affidabile. Deve quindi effettuare la frammentazione dei dati, la correzione degli errori e la prevenzione della congestione della rete. Il livello trascura la topologia della rete (end-to-end)
- 5-sessione** organizza il dialogo tra due programmi applicativi, consentendo di aggiungere a connessioni end-to-end servizi più avanzati
- 6-presentazione** definisce formalmente i dati che gli applicativi si scambiano, come questi dati sono rappresentati localmente sul sistema, e come vengono codificati durante il trasferimento.
- 7-applicazione** protocolli dei programmi applicativi, facenti parte del sistema operativo oppure scritti dall'utente, attraverso i quali l'utente utilizza la rete.

Incapsulamento



Intermediari



Intermediari — esempi

repeater livello 1, amplifica il segnale, es hub

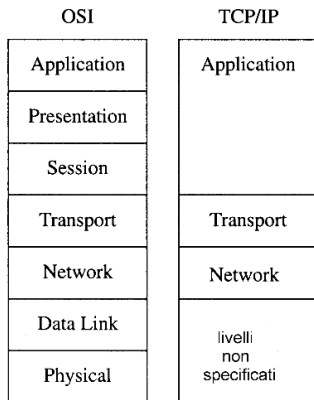
bridge livello 2, es switch

router livello 3

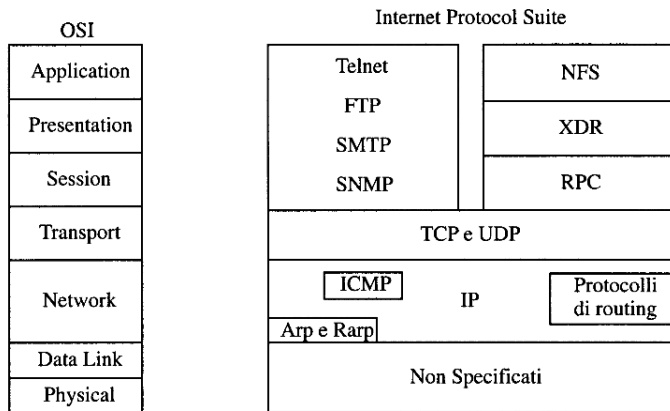
gateway livello 7, es proxy HTTP

Internet protocol Suite — TCP/IP

- ▶ nato in ambito UNIX
- ▶ lo standard de facto per la comunicazione su internet



Internet protocol Suite — esempi



Il protocollo TCP

Il servizio offerto da TCP sopra IP è il trasporto di un flusso di byte bidirezionale.

- ▶ È un protocollo orientato alla connessione, ovvero prima della trasmissione dei dati deve essere negoziata una connessione tra le due parti.
- ▶ Garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine e una volta sola. Questo è realizzato attraverso vari meccanismi di acknowledgement e di ritrasmissione su timeout.
- ▶ Possiede funzionalità di controllo delle congestioni di rete, attraverso il meccanismo denominato sliding window.
- ▶ Fornisce un servizio di multiplexing di molteplici connessioni sullo stesso host, attraverso il meccanismo delle porte.

Il protocollo UDP

- ▶ Non gestisce il riordino e la ritrasmissione dei pacchetti.
- ▶ Non è orientato alla connessione
- ▶ I dati inviati possono quindi essere persi
- ▶ È (per questi motivi) estremamente leggero, adatto a fare streaming
- ▶ Fornisce un servizio di multiplexing di molteplici connessioni sullo stesso host, attraverso il meccanismo delle porte.

Identificazione delle parti in gioco

In una rete basata su IP (versione 4)

- ▶ Ogni computer della rete identificato da un indirizzo IP unico a 32 bit (Es. 130.136.32.1).
- ▶ Gli indirizzi IPv4 sono difficili da ricordare (quelli v6 ancora peggio), quindi si usano dei nomi (Es. `www.google.com`) suddivisi in gerarchie (livelli).
- ▶ Server appositi, chiamati DNS(Domain Name System), offrono (anche) il servizio di conversione di nomi in indirizzi. (Es. `www.google.it` \mapsto 209.85.135.147)
- ▶ Per verificare l'associazione fra nomi e indirizzi si può utilizzare il comando `host`. (Es. `host www.google.it`)

Servizi e Server

Un server è composto da hardware e da software.

- ▶ Ogni server che eroga un servizio è collegato a una rete (Es. internet) e ha un indirizzo (Es. 130.136.1.110)
- ▶ Ad ogni servizio è associato un numero di porta univoco, in modo che lo stesso server possa erogare più servizi: il servizio è identificato dall'indirizzo del server e dalla porta
- ▶ I numeri di porta inferiori a 1024 sono riservati per i servizi di sistema (vedi `/etc/services` in Unix)
- ▶ Ai servizi erogati dal server corrispondono dei software in esecuzione (processi) in “ascolto” sulla porta corrispondente al servizio

Esempi di servizi e porte relative:

80 HTTP

21 FTP

22 SSH

Comunicazione Client-Server

Il protocollo IP (livello OSI rete) non prevede il concetto di porta, ma il protocollo TCP (livello OSI trasporto) sì.

- ▶ Un servizio offerto sopra il protocollo TCP è quindi identificato da $\langle \text{indirizzo IP server, porta servizio} \rangle$
- ▶ Il client stesso può effettuare più richieste di un servizio (anche a server differenti), quindi anche la sorgente della richiesta è identificata da $\langle \text{indirizzo IP client, porta processo client} \rangle$
- ▶ La comunicazione tra un Client e un Server può quindi essere descritta da una quintupla $\langle \text{protocollo, indirizzo IP server, porta server, indirizzo IP client, porta processo client} \rangle$

Esempio: $\langle \text{TCP, 123.23.4.221, 1500, 234.151.124.2, 4000} \rangle$

Concetto di socket (1/2)

Socket (spina) è l'astrazione che si usa per descrivere un canale di comunicazione.

- ▶ La quintupla vista in precedenza viene suddivisa nelle due componenti simmetriche (endpoint del canale) che costituiscono una associazione:

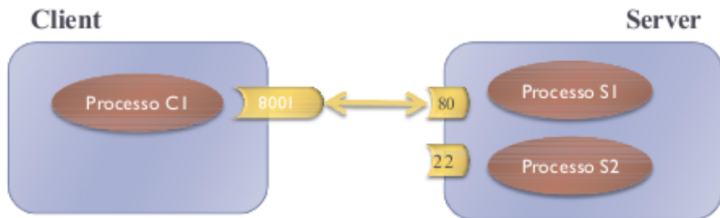
remota \langle protocollo, indirizzo IP remoto, porta remota \rangle

locale \langle protocollo, indirizzo IP locale, porta locale \rangle

Nota: esistono vari protocolli di livello trasporto (Es. UDP), e quindi il socket:

\langle TCP, 130.136.1.110, 53 $\rangle \neq \langle$ UDP, 130.136.1.110, 53 \rangle

Concetto di socket (2/2)



- ▶ Socket non è solo un concetto astratto, ma una vera e propria API (ideata a Berkley nel 1981) che consente di comunicare con processi in esecuzione su host (anche, ma non solo) remoti
- ▶ Useremo i socket per comunicare usando il protocollo di trasporto TCP

Creazione di un socket

Le seguenti operazioni costituiscono la vita di un socket.

Lato server

1. Creazione del socket
2. Bind ad una porta
3. Listen, predisposizione a ricevere sulla porta
4. Accept, blocca il server in attesa di una connessione
5. Lettura - scrittura dei dati
6. Chiusura

Lato client

1. Creazione del socket
2. Richiesta di connessione
3. Lettura - scrittura dei dati
4. Chiusura

Java socket

L'interfaccia ai socket in Java rispecchia le API ideate a Berkley, ma è orientata agli oggetti.

- ▶ Implementata nel package `java.net`
- ▶ Indirizzamento: `InetAddress`
- ▶ Connessioni TCP: `Socket`, `ServerSocket`
- ▶ Pacchetti UDP: `DatagramPacket`, `DatagramSocket`

Classe InetAddress

Rappresenta un indirizzo IP (sia versione 4 che 6). Per creare un oggetto di tale classe:

```
static InetAddress[] getAllByName(String host);
static InetAddress  getByAddress(byte[] addr);
static InetAddress  getByAddress(String host, byte[] addr);
static InetAddress  getByName(String host);
static InetAddress  getLocalHost();
```

Metodi interessanti:

```
String  getHostAddress() ;           /* pretty print */
String  getHostName();              /* eventuale lookup inverso */
boolean isReachable(int timeout); /* ping */
```

Esempio InetAddress

```
import java.net.*;
public class InetAddressTest {
    public static void main(String args[]){
        String hostname = "www.unibo.it";
        try {

            InetAddress ind = InetAddress.getByName(hostname);
            System.out.println("L'indirizzo IP di "+hostname+
                ": "+ind.getHostAddress());

        } catch(UnknownHostException e) {
            System.out.println("Impossibile risolvere "+hostname);
        }
    }
}
```

Classi per TCP: (Server)Socket

Java fornisce due diverse classi per la comunicazione con il protocollo TCP che rispecchiano la struttura client/ server:

ServerSocket Creazione socket per il server

Socket Creazione socket per il client

La differenziazione del socket Client e Server dovuto alle diverse operazioni che vengono svolte al momento di stabilire una connessione

- ▶ Il server ottiene un oggetto socket da una chiamata al metodo `accept` (uno diverso per ogni connessione instaurata)
- ▶ Il client deve provvedere a creare un'istanza del socket e stabilire la connessione

Classe: ServerSocket

Costruttori:

```
ServerSocket();  
ServerSocket(int port);  
ServerSocket(int port, int backlog); /* default 50 */  
ServerSocket(int port, int backlog, InetAddress bindAddr);
```

Tali costruttori comprimono in un'unica azione le operazioni di bind e listen. Altri metodi di interesse:

```
Socket accept();  
void setPerformancePreferences(  
    int connectionTime, int latency, int bandwidth);  
void setReceiveBufferSize(int size);  
void setSoTimeout(int timeout);  
void close();
```

Classe: Socket

Costruttori interessanti:

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

Tali costruttori effettuano la connessione oltre che creare il socket. Altri metodi di interesse:

```
InetAddress getLocalAddress();  
int getLocalPort();  
InputStream getInputStream();  
OutputStream getOutputStream();  
void close();
```

InputStream

InputStream è una classe abbastanza povera e complessa da usare, ad esempio utilizza bytes e non caratteri.

Ecco i metodi principali.

```
int available()  
int read(byte[] b, int off, int len)
```

```
boolean markSupported()  
void mark(int readlimit)
```

InputStreamReader

InputStreamReader aggiunge a InputStream funzionalità di decodifica di gruppi di bytes in caratteri.

Ecco i costruttori principali.

```
InputStreamReader(InputStream in)
```

```
InputStreamReader(InputStream in, Charset cs)
```

Ecco i metodi principali.

```
int read(char[] cbuf, int offset, int length)
```

BufferedReader

BufferedReader è una classe di alto livello.

Permette di incrementare le performance per mezzo di buffering.

Fornisce una funzione particolarmente utile.

```
BufferedReader(Reader in)
```

```
BufferedReader(Reader in, int sz)
```

```
String readLine()
```

OutputStream

OutputStream è una classe di basso livello e complessa da usare, ad esempio lavora con bytes e non caratteri.

Fornisce i seguenti metodi interessanti

```
void flush()  
void write(byte[] b)  
void write(byte[] b, int off, int len)
```

OutputStreamWriter

OutputStreamWriter aggiunge a OutputStream funzionalità di decodifica di gruppi di bytes in caratteri.

Ecco i costruttori principali.

```
OutputStreamWriter(OutputStream out)  
OutputStreamWriter(OutputStream out, Charset cs)
```

Ecco i metodi principali.

```
void write(char[] cbuf, int off, int len)
```

PrintWriter

PrintWriter è una classe di alto livello che permette di formattare in modo agevole una stringa prima di scriverla con il Writer sottostante.

Ecco i costruttori principali.

```
PrintWriter(Writer out)
```

```
PrintWriter(Writer out, boolean autoFlush)
```

Ecco i metodi principali.

```
PrintWriter printf(String format, Object... args)
```

```
void println(String x)
```

Esempio client (1/4)

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

public class ClientExample {
    public static void main(String args[]){

        int port = 4321;
        String address = "localhost";
        Socket s = null;

        try {
            /* connessione */
            s = new Socket(address,port);
            System.out.println("Connesso a " +
                s.getInetAddress() + ":"+s.getPort());

            InputStream sin = s.getInputStream();
            OutputStream sout = s.getOutputStream();
```

Esempio client (2/4)

```
/* preparazione stream dal server */
BufferedReader fromServer =
    new BufferedReader(new InputStreamReader(sin));

/* preparazione stream verso il server */
PrintWriter toServer =
    new PrintWriter(new OutputStreamWriter(sout));

/* preparazione stream dall'utente */
BufferedReader in =
    new BufferedReader(new InputStreamReader(System.in));
```

Esempio client (3/4)

```
String line;
/* Leggo stringa da tastiera */
System.out.print(">");
line = null; line = in.readLine();
if(line.equals("")) throw new IOException();

/* Invio stringa al server */
toServer.println(line);
toServer.flush();

/* Aspetto risposta dal server */
line = fromServer.readLine();
if(line == null) throw new IOException();

/* Visualizzo la risposta del server */
System.out.println(line);
```

Esempio client (4/4)

```
}catch (IOException e){
    System.err.println(e);
}
/* In ogni caso chiudo il socket (se aperto) */
finally{
    try{
        if(s!= null) s.close();
    }
    catch(IOException e2){
        /* Viene stampato lo stack */
        e2.printStackTrace();
    }
}
} /* Fine main */
} /* Fine classe */
```

Esempio server (1/3)

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

public class ServerIterativo {
    public static String reverse(String line) {
        StringBuffer revline;
        int len;
        len = line.length();
        revline = new StringBuffer(len);
        for(int i = len-1; i>=0; i--)
            revline.insert(len-1-i,line.charAt(i));
        return revline.toString();
    }
    public static void main(String[] args){
        String line;
        Socket conn = null;

        try{
            ServerSocket listen_socket = new ServerSocket(4321);
```

Esempio server (2/3)

```
while(true) {
    /* Aspetta un richiesta di connessione */
    conn = listen_socket.accept();

    /* stream di input e output */
    InputStream sin = conn.getInputStream();
    BufferedReader in =
        new BufferedReader(new InputStreamReader(sin));
    PrintStream out =
        new PrintStream(conn.getOutputStream());

    /* leggi dati inviati dal client */
    line = in.readLine();
    if(line != null) {
        out.println(reverse(line));
    }

    /* Chiudi connessione */
    conn.close();
}
```

Esempio server (3/3)

```
    } catch(Exception e){
        System.err.println(e);
    }
    /* Chiudi il socket */
    finally{
        try{
            if(conn!= null) conn.close();
        }
        catch(IOException e2){
            e2.printStackTrace();
        }
    }
} /* fine main */
} /* fine classe */
```

Classe DatagramPacket

È una classe che descrive i pacchetti (da ricevere o inviare)

```
public DatagramPacket(byte[] buf, int length);  
public DatagramPacket(byte[] buf, int length, InetAddress address
```

I metodi interessanti sono:

```
int getLength();  
InetAddress getAddress();
```

Classe DatagramSocket

La classe prevede il seguente costruttore (e altri meno interessanti):

```
DatagramSocket(int port);
```

I metodi interessanti sono:

```
void receive(DatagramPacket p);  
void send(DatagramPacket p);
```

I metodi meno interessanti sono:

```
public void connect(InetAddress address, int port)  
public DatagramChannel getChannel() /* mutex */
```

Esempio client datagram

```
import java.net.*; import java.io.*;
public class ClientDatagram {
    public static void main(String args[]){
        try{
            InetAddress dest = InetAddress.getByName("localhost");
            int port=8828;

            /* Messaggio da inviare */
            String msg="Ciao, come va?";
            byte[] data = msg.getBytes();

            /* Creo il pacchetto Datagram */
            DatagramPacket p =
                new DatagramPacket(data,data.length,dest,port);

            /* invio */
            DatagramSocket s = new DatagramSocket();
            s.send(p);

        } catch(Exception e){ e.printStackTrace(); } } }
```

Esempio server datagram

```
import java.net.*; import java.io.*;
public class ServerDatagram{
    public static void main(String args[]){
        try{

            String messaggio;

            DatagramSocket srv = new DatagramSocket(8828);
            DatagramPacket p = new DatagramPacket(new byte[1000],1000)

            while(true){

                // Ricevo il pacchetto
                srv.receive(p);
                messaggio =
                    new String(p.getData()).substring(0, p.getLength());
                System.out.println(messaggio);
            }
        } catch(Exception e){ e.printStackTrace(); } } }
```

Fine!

Esercizio (che vi consiglio di fare ora):

- ▶ andate in laboratorio
- ▶ scaricate `http://www.cs.unibo.it/~tassi/LSO-IPM/esempi-socket.zip`
- ▶ modificate il server concorrente in modo che accetti argomenti da linea di comando
- ▶ modificare il server concorrente in modo che si metta in ascolto su una porta specificata a linea di comando
- ▶ ...