



LABORATORIO di Reti di Calcolatori

Socket in linguaggio Java: modelli di servizio

Bibliografia

- ❖ slide della docente
- ❖ *testo di supporto*: D. Maggiorini, “Introduzione alla programmazione client-server”, Pearson Ed., 2009
 - ❑ cap.7 (tutto)
 - ❑ cap.8 (tutto)
- ❖ *Link utili*:
 - ❑ <http://docs.oracle.com/javase/tutorial/networking/index.html>
 - ❑ <http://docs.oracle.com/javase/6/docs/>

...e i modelli di servizio?!

	connection-oriented	connectionless
s. iterativo	✓	non ha senso
s. concorrente	non supportato dal linguaggio	non supportato dal linguaggio <i>(ma viene gratis)</i>
s. multi-thread	✓	non ha senso

- ❖ in realtà, per servizio connection-less non ha senso parlare di alcun modello di servizio...
- ❖ server iterativo: si ottiene facilmente da esempi mostrati

```
while(true) {
    accept;
    comunicazione con client corrente su nuova socket;
    chiusura socket dedicata;
}
```

→ homework!

pseudo-codice client/server iterativo

CLIENT

```
do {
    letto ← read from tastiera;
    toServer.write(letto);
    if (letto != carattere finale)
        toServer.read(buffer);
} while(letto != carattere finale);
toServer.close();
```

SERVER

```
while(true)
{
    fromClient ← ServSock.accept();
    do {
        fromClient.read(letto);
        if (letto != carattere finale)
            fromClient.write(letto);
    } while((letto != carattere finale);
    fromClient.close();
}
ServSock.close();
```

server connection-oriented concorrente

- ❖ si può fare con **time-out** e politica di polling (analisi circolare su socket passiva, e tutte le socket attive già aperte)
- ❖ **void setSoTimeout(int msec)**
 - ❑ su ServerSocket interrompe accept
 - ❑ su Socket interrompe read
 - ❑ se scatta timeout è sollevata eccezione
 - ma socket restano valide
 - ❑ se msec=0 → attesa infinita
- ❖ è un **busy waiting (!)**
- ❖ **problema gestione socket client chiuse**

```

Socket[] fromcl;
serverSocket passive;
int index=0;
while(True)
{
    try{
        passive.setSoTimeout(3);
        fromcl[index] ← passive.accept();
        index++;
    } catch( java.net.SocketTimeoutException){...}
    for(i=0; i++; i<index)
    {
        fromcl[i].setSoTimeout(3);
        try{
            fromcl[i].read();
            consuma quanto letto;
        } catch(java.net.SocketTimeoutException)
        {...}
    }
} // end for
} // end while

```

esempio codice (1)

il server alterna il suo tempo tra guardare la socket passiva per nuove richieste di connessione, e guardare le socket client

```

44      sServ = new ServerSocket(0);
45      System.out.println("sAddr: " + sServ.getInetAddress()
46                          + " "; sPort: " + sServ.getLocalPort());
47      while(true){
48          // Creazione ServerSocket
49          // Accept Client fino a Timeout o max_conn
50          try{
51              sServ.setSoTimeout(sServ_timeout);
52
53              while(index<max_conn){
54                  sClient.add(sServ.accept());
55                  index++;
56              }
57          } catch(SocketTimeoutException ste){
58              System.out.println("\nServerSocket: Timeout expired!!!\n");
59          } catch(IOException ioe){
60              System.out.println("SocketServer Exception:");
61              ioe.printStackTrace();
62          }

```

ArrayList<Socket> sClient =
new ArrayList<Socket>(max_conn);

```

65 // Gestione RR dei Client
66 while(index > 0){
67     System.out.println("cAddr: " + sClient.get(i).getInetAddress()
68         + "; cPort: " + sClient.get(i).getPort());
69     try{
70         sClient.get(i).setSoTimeout(sClient_timeout);
71         InputStream isC = sClient.get(i).getInputStream();
72         while(true){
73             int letti = isC.read(buff);
74             String str_cli = (new String(buff, 0, letti)).trim();
75
76             if(str_cli.equals("")){
77                 //throw new Exception("End of Client");
78                 sClient.get(i).close();
79                 sClient.remove(i);
80                 index--;
81                 break;
82             }
83             System.out.println(str_cli);
84         }
85     }catch(SocketTimeoutException ste){
86         System.out.println("Client: Timeout expired!!!");
87     }catch(Exception e){
88         e.printStackTrace();
89         try{
90             sClient.get(i).close();
91             sClient.remove(i);
92             index--;
93         }catch(IOException ioe){
94             ioe.printStackTrace();
95         }
96     }
97     i = index!=0 ? (i+1)%index : 0;
98 }

```

parte (2)

Elena Pagani
LABORATORIO Reti di Calcolatori – A.A. 2019/2020
7 of 14

server multi-thread

- ❖ isoliamo la parte di comunicazione con il cliente in una classe che estende la classe Thread
- ❖ il metodo `run` di tale nuova classe deve eseguire la parte di codice che gestisce la comunicazione con il client
- ❖ in alternativa:
 - ❑ sul thread viene chiamato il metodo `start` dopo la creazione
 - ❑ oppure, il metodo `start` è inglobato nel creatore della nuova classe
 - ❑ ... teniamo il client come nel primo esempio di servizio connection-oriented
 - N.B.: bisogna ricompilare con porta server corretta

Elena Pagani
LABORATORIO Reti di Calcolatori – A.A. 2019/2020
8 of 14

server multi-thread

```

9 public class es1SrvIter
10 {
11     public static void main(String[] args)
12     {
13         ServerSocket sSrv;
14         Socket toClient;
15         try {
16             sSrv = new ServerSocket(0);
17             System.out.println("Indirizzo: " + sSrv.getInetAddress()
18                             + "; porta: " + sSrv.getLocalPort());
19
20             while (true)
21             {
22                 toClient = sSrv.accept();
23                 System.out.println("Ind Client: " + toClient.getInetAddress()
24                                 + "; porta: " + toClient.getPort());
25                 Thread t = new erogServizio(toClient);
26                 t.start();
27                 // toClient.close();
28             }
29         } catch (Exception e) {
30             e.printStackTrace();
31         }
32     }
33 }

```

problema: la socket connessa al client va chiusa solo quando termina la comunicazione con esso, ovvero quando termina il thread (fig.8.23)

Elena Pagani

LABORATORIO Reti di Calcolatori – A.A. 2019/2020

9 of 14

classe per server dedicato

```

6 public class erogServizio extends Thread
7 {
8     private Socket sock2Cl;
9
10    public erogServizio(Socket socket)
11    {
12        this.sock2Cl = socket;
13    }
14
15    public void run()
16    {
17        int dim_buffer = 100;
18        byte buffer[] = new byte[dim_buffer];
19
20        while (true)
21        {
22            try {
23                InputStream fromCl = sock2Cl.getInputStream();
24                int letti = fromCl.read(buffer);
25                if (letti > 0) {
26                    String stampa = new String(buffer, 0, letti);
27                    System.out.println("Ricevuta stringa: " + stampa + " di " + letti + " byte da " + sock2Cl.getInetAddress()
28                                    + "; " + sock2Cl.getPort());
29                }
30                else {
31                    sock2Cl.close();
32                    return;
33                }
34            } catch (Exception e) {
35                e.printStackTrace();
36            }
37        }
38    }
39 }

```

eseguito quando si fa partire (start) il thread

stampiamo identità client da cui si è ricevuto lo specifico messaggio

qui si!
e si chiude anche per server primario perché condividono memoria

- ❖ provare a lanciare più client concorrenti da terminali differenti
- ❖ il server (*giustamente*) non termina mai...

Elena Pagani

LABORATORIO Reti di Calcolatori – A.A. 2019/2020

10 of 14

homework

- ❖ modificare client/server connessi in modo che:
 1. il client possa mandare più stringhe. Il client termina quando riceve in input da tastiera il carattere ‘.’ → lo invia al server che chiude connessione con questo client
 2. il server invii in risposta al client la stringa da esso ricevuta (servizio standard *Echo*)
 3. punti 1+2 con server sia iterativo sia multi-thread che gestisce conversazioni con più client contemporaneamente
- ❖ modificare client/server connectionless in modo che:
 1. il server invii in risposta al client la stringa da esso ricevuta (servizio standard *Echo*)
 - guardando il file `/etc/services` si scopre che *Echo* è un servizio (standard) **multiprotocollo**: può usare sia UDP sia TCP
- ❖ testare i codici con più client contemporanei

alcune considerazioni finali

- ❖ il S.O. memorizza le richieste di connessione dei client in una coda first-in first-out
 - la massima lunghezza coda dipende dal S.O. (di solito 50)
 - costruttore `ServerSocket(int port, int backlog, InetAddress bindAddr)`
 - Se il numero di richieste in coda eccede la capacità massima, le successive richieste vengono **scartate** direttamente dal S.O.
 - Il client deve gestire le situazioni in cui la richiesta di connessione non va a buon fine
 - quindi: bisogna fare il *catch* dell’eccezione e gestirla

alcune considerazioni finali

- ❖ `ServerSocket.close()` rilascia la porta passiva e tutte le porte create da `accept()`
 - ❑ lo fa anche il garbage collector quando il programma termina
 - ❑ in ogni caso, le porte **non** sono immediatamente riutilizzabili
 - → Teoria per definizione *Maximum Segment Lifetime*
- ❖ Java permette anche limitata configurazione del modo di operazione delle socket
 - ❑ `Socket.getReuseAddress()` / `Socket.setReuseAddress()`
 - ❑ `Socket.setKeepAlive()` , `Socket.SoTimeout()`
 - ❑ ...fare riferimento alle lezioni di Teoria

gestione eccezioni

- ❖ negli esempi fatta un po' brutalmente
- ❖ bisognerebbe distinguere i vari casi di errore ed intraprendere operazioni opportune in dipendenza della semantica del servizio
 - ❑ quando il server è in situazione di errore e va chiuso?
 - ❑ quando la connessione è in situazione di errore e va chiusa, ma il server può continuare ad operare con altri client?
- ❖ distinguere tra errori su indirizzi, errori su canali, errori su I/O da tastiera...