LEZIONE N.5
LPR INFORMATICA APPLICATA
CONNECTION ORIENTED SOCKETS (2)

17/03/2008

Laura Ricci

STREAM MODE SOCKET API: INTEZIONE CON SERVERS PREDEFINITI

Esercizio: considerare un servizio attivo su una porta pubblicata da un Server.

Definire un client JAVA che utilizzi tale servizio. Il linguaggio con cui è scritto il client è indipendendente da quello con cui è scritto il server.

Si possono considerare i seguenti semplici servizi (vedere JAVA Network Programming)

Daytime(porta 13): il client richiede una connessione sulla porta 13, ma non invia alcun dato al server. Il server invia la data odierna e l'ora come una stringa di caratteri e quindi chiude la connessione.

Echo (port 7): il client apre una connesione sulla porta 7 del server ed invia un messaggio. Il server restituisce il messaggio al client. Utilizzato per testare e misurare lo stato della rete

STREAM MODE SOCKET API: INTEZIONE CON SERVERS PREDEFINITI

Finger (porta 79): il client si connette ad un server specificando un username oppure un indirizzo di posta. Il server restituisce informazioni su un utente specificato dal client (es:mail, n.telefono,n.stanza, etc.)

```
% finger ricci@di.unipi.it
```

Whois (porta 43): il client invia un'interrogazione per a quale provider Internet appartenga un determinato indirizzo IP o uno specifico DNS

Altri servizi noti:

- 22 SSH
- 25 SMTP
- 80 HTTP

ECHO CLIENT TCP

- Il client esegue iterativamente le seguenti operazioni
  - richiede una stringa in input
  - ne attende l'echo dal server

  fino a quando viene inserita la stringa 'exit', nel qual caso chiude la comunicazione e termina

ECHO CLIENT TCP

```
import java.net.*; import java.io.*; import java.util.*;
public class EchoClient {
public static void main (String args[ ]) throws Exception
   {   Scanner console = new Scanner( System.in);
       InetAddress ia=InetAddress.getByName("localhost");
      int port=7;    Socket echosocket=null;
      try{   echosocket = new Socket (ia, port);}
      catch (Exception e){System.out.println(e);return;}
      InputStream is = echosocket.getInputStream( );
      DataInputStream NetworkIn = new DataInputStream(is);
      OutputStream os=echosocket.getOutputStream();
      DataOutputStream NetworkOut = new DataOutputStream(os);
```

ECHO CLIENT TCP

```
boolean done=false;
while (! done)
   {String linea = console.nextLine( );
    System.out.println (linea);
    if  (linea.equals("exit")) {NetworkOut.writeUTF(linea);
                                NetworkOut.flush( ); done = true;
                                echosocket.close ( ); }
          else     {NetworkOut.writeUTF (linea);
                                NetworkOut.flush( );
                                String echo=NetworkIn.readUTF( );
                                System.out.println (echo);
                               } } } }
```

ECHO SERVER TCP

Echo Server

- si mette in attesa di richieste di connessione
- dopo aver accettato una connessione, si mette in attesa di una stringa dal client e gliela rispedisce
- quando riceve la stringa 'exit' chiude la connessione con quel client e torna ad accettare nuove connessioni

ECHO SERVER TCP

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

public class EchoServer {

    public static void main (String args[ ]) throws Exception{

    int port=.....;
    ServerSocket ss= new ServerSocket(port);
```

ECHO SERVER TCP

```
while (true)
   {Socket sdati = ss.accept( );
    InputStream is = sdati.getInputStream( );
    DataInputStream networkIn = new DataInputStream(is);
    OutputStream out=sdati.getOutputStream( );
    DataOutputStream networkOut = new DataOutputStream(out);
     boolean done=false;
     while (!done){
       String echo= networkIn.readUTF( );
       if (echo.equals("exit"))
           {System.out.println("finito");  done=true;}
       else {networkOut.writeUTF(echo);} } }  } }
```

SOCKETS LATO CLIENT: CHIUSURA

- Un socket (oggetto della classe Socket) viene chiuso automaticamente a causa di:
  - garbage collection
  - terminazione del programma
  - chiusura di uno degli streams associati
- In certi casi (esempio un web browser)
  - il numero di sockets aperti può essere molto alto
  - il numero massimo di sockets supportati può essere raggiunto prima della garbage collection
  - può essere necessario chiudere esplicitamente alcuni sockets che non vengono più utilizzati
  - chiusura esplicita di un socket s : s.close( )

CHIUSURA DI SOCKETS

```
for (int i = 1; i< 1024; i++)
   {try
        { s = new Socket(host, i);
             System.out.println("Esiste un servizio sulla porta"+i); }
   catch (UnknownHostException ex)
        {System.out.println("Host  Sconosciuto"); break;}
   catch (IOException ex) {System.out.println("Non esiste un servizio
                                                sulla porta"+i);}
   finally{  // la clausola finally viene eseguita dopo il blocco try...catch,
             // qualunque sia il modo con cui si è usciti da tale blocco
     try{
        if (s!=null)     {s.close( ); s=null;  System.out.println("chiuso");}
        } catch(IOException ex){   };      }     }
```

INVIO DI OGGETTI SU CONNESSIONI TCP

- Per inviare oggetti su una connessione TCP occorre definire l'oggetto come istanza di una classe che implementa l'interfaccia Serializable
- E' possibile associare i filtri ObjectInputStream/ ObjectOutputStream agli stream di bytes associati al socket e restituiti da getInputStream/getOutputStream
- Quando creo un ObjectOutputstream viene scritto lo stream header sullo stream. In seguito scrivo gli oggetti che voglio inviare sullo stream
- L'header viene scritto una sola volta quando lo stream viene creato e viene letto alla prima lettura sullo stream
- L'invio/ ricezioni degli oggetti sullo/dallo stream avviene mediante scritture/letture sullo stream (writeObject, readObject)

INVIO DI OGGETTI SU UNA CONNESSIONE TCP

```
import java.io.*;
    public class Studente implements Serializable {
    private int matricola;
    private String nome, cognome, corsoDiLaurea;
    public Studente (int matricola, String nome, String cognome,
                        String corsoDiLaurea) {
          this.matricola = matricola; this.nome = nome;
    this.cognome = cognome; this.corsoDiLaurea = corsoDiLaurea;}
    public int getMatricola () { return matricola; }
    public String getNome () { return nome; }
    public String getCognome () { return cognome; }
    public String getCorsoDiLaurea () { return corsoDiLaurea; }   }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO SERVER

```
import java.io.*; import java.net.*;
public class Server {
public static void main (String args[]) {
try { ServerSocket server = new ServerSocket (3575);
      Socket clientsocket = server.accept();
      ObjectOutputStream output =
                  new ObjectOutputStream (clientsocket.getOutputStream ());
      output.writeObject("<Welcome>");
      Studente studente = new Studente (14520,"Mario","Rosso","Informatica");
      output.writeObject(studente);  output.writeObject("<Goodbye>");
      clientsocket.close();
      server.close(); } catch (Exception e) { System.err.println (e); }  } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP-LATO CLIENT

```
import java.io.*; import java.net.*;
public class Client { public static void main (String args[ ]) {
try { Socket socket = new Socket ("localhost",3575);
ObjectInputStream input =
                new ObjectInputStream (socket.getInputStream ());
String beginMessage = (String) input.readObject();
System.out.print (studente.getMatricola()+" - ");
System.out.print (studente.getNome()+" "+studente.getCognome()+" - ");
System.out.print (studente.getCorsoDiLaurea()+"\n");
String endMessage = (String)input.readObject();
System.out.println (endMessage); socket.close();} catch (Exception e)
   { System.out.println (e); } } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO CLIENT

Stampa prodotta lato Client

<Welcome>

14520 - Mario Rossi - Informatica

<Goodbye>

CLASSE SOCKET: OPZIONI

- la classe socket offre la possibilità di impostare diverse proprietà (opzioni)
- ogni opzione consente di controllare il modo con cui i dati vengono scabiati attraverso il socket
- Opzioni
  - SO_TIMEOUT
  - SO_LINGER
  - SO_KEEPALIVE
  - SO_RCVBUF
  - SO_SNDBUF
  - TCP_NODELAY

  .....

CLASSE SOCKET: SO_TIMEOUT

SO_TIMEOUT – consente di associare un time out al socket

```
if (s.getSoTimeout( )== 0) s.setSoTimeout(1800000);
```

- Quando eseguo una lettura da uno stream socket, l'operazione si blocca fino al momento in cui ci sono byte sufficienti per restituire il dato richiesto
- SO_TIMEOUT: definisce un intervallo di tempo massimo per l'attesa dei dati

CLASSE SOCKET: SO_LINGER

La proprietà SO_LINGER (LINGER = indugiare,soffermarsi) viene utilizzata per specificare cosa accade quando viene invocato il metodo close( ) su un socket TCP

A seconda del valore di SO_LINGER può accadere che

- Linger= false (default): il contenuto del buffer di invio associato al socket viene inviato al destinatario, mentre i dati nel buffer di ricezione vengono scartati. Il metodo close( ) non attende la terminazione di queste attività che avvengono quindi in background.

  Questo è lo scenario di default, che però non garantisce che i dati vengano consegnati correttamente. In caso di crash del destinatario, ad esempio, i dati nel buffer di spedizione non vengono consegnati

CLASSE SOCKET: SO_LINGER

- Linger=true, Linger time=0 Vengono scartati sia gli eventuali dati nel buffer di ricezione che quelli da inviare. Come prima, lo scarto avviene in background
  - Utilizzato quando si vuole terminare la connessione immediatamente, senza spedire i dati
- Linger=true e linger time!=0 Vengono inviati eventuali dati presenti nel buffer al destinatario e si scartano gli eventuali dati nel buffer di ricezione. Il metodo close( ) si blocca per il linger time oppure fino a che tutti i dati spediti sono stati confermati a livello TCP. Dopo linger time viene sollevata un'eccezione
  - Quando si vuole garantire che il metodo close() ritorni solo quando i dati sono stati consegnati, oppure che sollevi un'eccezione nel caso in cui scatti il time-out definito da linger-time

CLASSE SOCKET:SO_LINGER

```
public void setSoLinger (boolean no, int seconds)
                                              throws SocketException
public int getSoLinger ( ) throws SocketException
```

- per default, SO_LINGER=false: il supporto tenta di inviare i datagrams rimanenti, anche dopo che il socket è stato chiuso
- per controllare la gestione dei datagrams presenti al momento della chiusura

```
if (s.getSoLinger( )== -1) s.setSoLinger(true,240);
```

il metodo close( ) si blocca ed attende 240 secondi (4 minuti) prima di eliminare i datagrams rimanenti. Se il tempo di attesa viene impostato a 0, i datagram vengono eliminati immediatamente.

CLASSE SOCKET: SO_KEEPALIVE

- So_keepalive: introdotto per individuare i sockets "idle" su cui non sono stati inviati dati per un lungo intervallo di tempo
- Per default, su ogni socket vengono spediti solo dati inviati dalla applicazione
- Un socket può rimanere inattivo per ore, o anche per giorni
- Esempio: crash di un client prima dell'invio di un segnale di fine sequenza. In questo caso, il server può sprecare risorse (tempo di CPU, memoria,...) per un client che ha subito un crash
- someSocket.setSoKeepAlive(true) abilita il keep alive.
  - Il supporto invia periodicamente dei messaggi di keep alive sul socket per testare lo stato del partner.
  - Se il partner non risponde, il socket viene chiuso automaticamente
- alternativa all'uso di time-out associati al socket

CLASSE SOCKET: SO_RCVBUF, SO_SNDBUF

- SO_RCVBUF Controlla la dimensione del buffer utilizzato per ricevere i dati.
  - E' possibile impostare la dimensione del buffer di ricezione

    Socket.setReceiveBufferSize(4096)
  - La modifica non viene garantita su tutti i sistemi operativi
  - Per reperire la dimensione del buffer associato

    int size = sock. getReceiveBufferSize( )
  - Alternativa: utilizzare i BufferedInputStream/BufferedReader.

- SO_SNDBUF : analogo per il buffer associato alla spedizione

  int size = sock.getSendBufferSize( );

CLASSE SOCKET: TCP_NODELAY

Algoritmo di Nagle:

- riduce il numero di pacchetti spediti sulla rete fondendo in un unico pacchetto più dati
- può interagire in modo scorretto con il meccanismo dei delayed ack
- Il TCP_NODELAY controlla l'uso dell'algoritmo di bufferizzazione di Nagle

  sock.setTcpNoDelay(false)

  disabilita/abilita la bufferizzazione
- Alcuni sistemi operativi disabilitano l'algoritmo di Nagle di default

ESERCIZIO

Sviluppare un'applicazione che offra il servizio di trasferimento di RemoteCopyClient ad un server RemoteCopyServer. Il client richiede, in modo interattivo, il nome del file da trasferire all'utente, e, se il file esiste, richiede una connessione al Server. Quando la connessione viene accettata, invia al server il nome del file seguito dal suo contenuto. Infine il client attende l'esito della operazione dal server, quindi torna a proporre una nuova richiesta di trasferimento all'utente.

RemoteCopyServer riceve una richiesta di connessione e salva il file richiesto in una directory locale. Alla fine del download del file, RemoteCopyServer invia al client l'esito della operazione. L'esito può essere di due tipi:

- update: il file esisteva già ed è stato sovrascritto
- new: è stato creato un nuovo file