Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA

Documentazione java

Programmazione e Sicurezza delle Reti

Riassunto delle principali librerie java
ed esempi di applicazioni

Autore:
Danzi Matteo
Matricola VR388529

Anno Accademico 2015/2016

Indice

1 java.net.*

Il pacchetto `java.net.*` fornisce classi e interfacce per l'implementazione di applicazioni di rete:

- Classi Socket e ServerSocket per le connessioni TCP
- Classe DatagramSocket per le connessioni UDP
- Classe URL per le connessioni HTTP
- Classe InetAddress per rappresentare gli indirizzi Internet
- Classe URLConnection per rappresentare le connessioni a un URL

1.1 Socket

```
public class Socket
extends Object
implements Closeable
```

Questa classe implementa un socket lato client per connessioni TCP.
Un socket è un canale bidirezionale di comunicazione tra due macchine. Il lavoro effettivo del socket è attuato da un metodo della classe `SocketImpl`.
Il costruttore principale di `Socket` che abbiamo usato è:

```
Socket(String host, int port) throws IOException,
            UnknownHostException
// crea un socket lato client, ricevendo:
// host = indirizzo IP(di tipo Stringa)
// port = porta(> 1024 e < 49151)
// esempio di creazione un socket lato client:
try{
        Socket s = new Socket("localhost", 11111);
}catch(IOException e){
        System.err.println(e.getMessage());
        System.exit(1);
}
```

Se l'host specificato è `null` è equivalente a inserire l'indirizzo di loopback della propria interfaccia("localhost").
Per chiudere il socket lato client si usa il metodo `close()`:

```
public void close()
          throws IOException
// l'utilizzo di questo metodo, come per
//  il costruttore, va incluso in un blocco try-catch.
```

Chiudendo il socket lato client si chiude automaticamente sia l'`InputStream` che l'`OutputStream` del socket.

1.2 ServerSocket

```
public class ServerSocket
extends Object
implements Closeable
```

Questa classe implementa un socket lato server per connessioni TCP.
Questo socket lato server attende la richiesta di apertura della connessione da parte di un client.
Solitamente esegue delle operazioni basate sulla richiesta e ritorna il risultato al richiedente.
Il costruttore principale di `ServerSocket` che abbiamo usato è:

```
ServerSocket(int port) throws IOException
// crea un socket lato server, ricevendo:
// port = porta su cui il server attende la richiesta
// esempio di creazione un socket lato server:
try{
        ServerSocket s = new ServerSocket(11111);
}catch(IOException e){
        System.err.println(e.getMessage());
        System.exit(1);
}
```

Per rendere effettiva la connessione una volta creata la socket abbiamo usato il metodo `accept()` il quale ascolta il canale per la connessione; blocca l'esecuzione fino al momento in cui riceve una richiesta di connessione.

```
public Socket accept() throws IOException
//esempio:
try{
        Socket clientSocket = s.accept();
}catch(IOException e){
        System.err.println("Connessione fallita");
        System.exit(2);
}
```

Per chiudere la connessione si usa il metodo `close()` della classe `Socket`.

1.3 DatagramSocket

```
public class DatagramSocket
extends Object
implements Closeable
```

Questa classe implementa un socket per inviare e ricevere pacchetti per connessioni UDP.
Questo DatagramSocket è il punto di ricezione/invio di un servizio di consegna di pacchetti. Ogni pacchetto ricevuto/inviato in un DatagramSocket è indirizzato e instradato singolarmente. Se ci sono più pacchetti da inviare, l'ordine di arrivo di questo può non coincidere con l'ordine di invio.
Costruttore:

```
public DatagramSocket() throws SocketException
```

1.4 URL

Questa classe crea un oggetto che rappresenta il Uniform Resource Locator, cioè un puntatore ad una risorsa sul Web. Questa risorsa può essere un semplice file o cartella, oppure può essere un riferimento a un oggetto più complicato come una query su un data base o su un motore di ricerca. Costruttore:

```
public URL(String spec) throws MalformedURLException
// spec = indirizzo corrispondente in formato URL
```

1.5 InetAddress

Questa classe rappresenta un indirizzo IP (Internet Protocol). Un'istanza di un oggetto di tipo InetAddress consiste in un indirizzo IP e possibilmente il nome dell'host corrispondente.

```
public class InetAddress
extends Object
implements Serializable
```

Il metodo `getByName(String host)` resituisce l'indirizzo IP corrispondente al nome dell'host inserito come argomento.

1.6 URLConnection

Questa classe astratta è la superclasse di tutte le classi che rappresentano una comunicazione tra l'applicazione e un URL. I metodi di questa classe possono essere usati sia per leggere che per scrivere sulla risorsa a cui fa riferimento l'URL. In generale creare una connessione ad un URL è un processo multistep.

```
public abstract class URLConnection
extends Object
```

Per creare una connessione ad un URL si usano i seguenti metodi:

```
URLConnection c = url.openConnection(); // url di tipo URL
c.connect();
```

1. L'oggetto connessione è generato invocando il metodo `openConnection` su un URL
2. La connessione effettiva all'oggetto remoto è fatta usando il metodo `connection()` sull'oggetto ritornato da `openConnection` di tipo `URLConnection`
3. L'oggetto remoto diventa disponiblie e i campi header e i contenuti dell'oggetto sono accessibili.

2 Applicazione UDP

```
//CLIENT UDP
import java.io.*;
import java.net.*;

public class UDPClient{
  public static void main(String[] args) throws Exception{
        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("localhost");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        System.out.println("Inserisci il messaggio per il Server");
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();

        DatagramPacket sendPacket =
          new DatagramPacket(
            sendData, sendData.length, IPAddress, 9876);
        clientSocket.send(sendPacket);

        DatagramPacket receivePacket =
          new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);

        String serverSentence = new String(receiveData.getData());
        System.out.println("From Server: " + serverSentence);

        clientSocket.close();
  }
}
//SERVER UDP
import java.io.*;
import java.net.*;

public class UDPServer{
  public static void main(String[] args) throws Exception{
    try{
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      while(true){
        try{
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);
                String sentence = new String(receivePacket.getData());
                InetAddress IPAddress = receivePacket.getAddress();
                int port = receivePacket.getPort();
                String answer = sentence + " ok";
                sendData = answer.getBytes();
                DatagramPacket sendPacket =
                  new DatagramPacket(
                    sendPacket, sendPacket.length, IPAddress, port);
                serverSocket.send(sendPacket);
              }catch(UnknownHostException ue){}
        }
        }catch(java.net.BindException b){}
  }
}
```

3 Chat TCP

```
//CLIENT TCP
import java.io.*;
import java.net.*;
import java.util.Scanner;

class Client{
  public void main(String[] args){
    try{
      Socket s = new Socket("localhost", 8000);
      PrintStream os = new PrintStream(
        new BufferedOutputStream(s.getOutputStream()));
    }catch(IOException e){
      System.err.println(e.getMessage());
      System.exit(1);
    }

    readerThread r = new readerThread(s);
    Thread thread = new Thread(r);
    thread.start();

    System.out.println("Socket creata: " + s);

    Scanner l = new Scanner(System.in);
    String out = null;
    while((out = l.nextLine()) != null){
          os.println(out);
          os.flush();
        }
        l.close();
        os.close();
        r.terminate();
  }
}
class readerThread implements Runnable{
  Socket s;
  boolean finish;
  readerThread(Socket s){
    this.s = s;
    this.finish = false;
  }

  public void terminate(){
    this.finish = true;
  }

  @Override
  public void run(){
    BufferedReader b = null;
    String in = null;
    try{
      b = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
      while(finish == false){
        in = new String(b.ReadLine());
        while(in != null)
          System.out.println(in);
      }
      b.close();
      s.close();
    }catch(IOException e){
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }
}
```

```java
//SERVER TCP
import java.io.*;
import java.net.*;
import java.util.List;
import java.util.ArrayList;

class Server{
  public static void main(String[] args) throws Exception{
        ServerSocket connection = new ServerSocket(8000);
        List<Server1> clients = new ArrayList<Server1>();
        while(true){
          Server1 s = new Server1(conn.accept(), clients);
          clients.add(s);
        }
  }
}
class Server1 implements Runnable{
  Socket s;
  List<Server1> clients;
  String nickname;
  public Server1(Socket s, List<Server1> clients){
    this.s = s;
    this.clients = clients;
    new Thread(this).start();
  }

  @Override
  public void run(){
        String from;
        try{
          in = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
          out = new PrintStream(s.getOutputStream());
          System.out.println("Connected " + s);
          out.println("Inserisci il nickname: ");

          do{
            this.nickname = in.readLine();
            if(isAlreadyUsed())
              out.println("Nickname gia in uso, reinserire: ");
            else
              out.println("ok");
          }while(isAlreadyUsed);

          while((from = in.readLine()) != null){
            transmitToAll(this.nickname + ": " + from , out);
            System.out.println(this.nickname + ": " + from);
          }

          transmitToAll(this.nickname + " ha abbandonato" , out);
          s.close();
          clients.remove(this);
        }catch(IOException e){}
        System.out.println(this.nickname + " "
          + this.getPort() + " Disconnected");
  }

  public void transmitToAll(String s, PrintStream out){
    try{
      for(Server1 ser : clients){
        out = new PrintStream((ser.s).getOutputStream());
        if(!this.equals(ser))
          out.println(s);
      }
    }catch(IOException e){}
  }
```

```
  public boolean isAlreadyUsed(){
    for(Server1 ser : clients){
      if(!this.equals(ser) && this.nickname.equals(ser.nickname))
        return true;
    }
    return false;
  }
}
```