Datagrammi

Le applicazioni che comunicano tramite socket possiedono un canale di comunicazione dedicato. Per comunicare, un client ed un server stabiliscono una connessione, trasmettono dati, quindi chiudono la connessione. Tutti i dati inviati sul canale sono ricevuti nello stesso ordine in cui sono stati inviati.

Le applicazioni che comunicano tramite *datagrammi* inviano e ricevono pacchetti di informazione completamente indipendenti tra loro. Queste applicazioni non dispongono e non necessitano di un canale di comunicazione punto-a-punto. La consegna dei *datagrammi* alla loro destinazione non è garantita, e neppure l'ordine del loro arrivo.

Il package java.net fornisce tre classi che consentono di scrivere applicazioni che usano datagrammi per inviare e ricevere pacchetti sulla rete: ***DatagramSocket***, ***DatagramPacket*** e ***MulticastSocket***. Una applicazione può inviare e ricevere *DatagramPacket* tramite un *DatagramSocket*. Inoltre, i *DatagramPacket* possono essere inviati in broadcast a più destinatari che sono in ascolto su un *MulticastSocket*.

NOTA: MulticastSocket estende DatagramSocket

java.net.DatagramSocket

- **DatagramSocket (int port)**
  Crea un DatagramSocket e lo collega alla porta specificata sulla macchina locale.
- **void receive (DatagramPacket p)**
  Riceve un DatagramPacket da questo socket.
- **void send (DatagramPacket p)**
  Invia un DatagramPacket su questo socket.
- **void close ()**
  Chiude questo DatagramSocket.

java.net.DatagramPacket

- **DatagramPacket (byte[] buf, int length)**
  Crea un DatagramPacket per ricevere pacchetti di lunghezza length.

- **DatagramPacket (byte[] buf, int length, InetAddress address, int port)**
  Crea un DatagramPacket per inviare pacchetti di lunghezza length all'host ed alla porta specificati.

- **InetAddress getAddress ()**
  Restituisce l'indirizzo IP della macchina alla quale questo DatagramPacket deve essere inviato o da cui è stato ricevuto.

- **int getPort ()**
  Restituisce la porta della macchina alla quale questo DatagramPacket deve essere inviato o da cui è stato ricevuto.

TimeClient

```
import java.io.*;
import java.net.*;
import java.util.*;
public class TimeClient {
   public static void main(String[] args) throws IOException {
          String hostname = "localhost";
          DatagramSocket socket = new DatagramSocket();
          // invia la richiesta
          byte[] buf = new byte[256];
          InetAddress address = InetAddress.getByName(hostname);
          DatagramPacket packet =
             new DatagramPacket(buf, buf.length, address, 3575);
          socket.send(packet);
           // riceve la risposta
          packet = new DatagramPacket(buf, buf.length);
          socket.receive(packet);
          // visualizza la risposta
          String received = new String(packet.getData());
          System.out.println("Response: " + received);
          socket.close();
    }
}
```

TimeServer (1)

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

public class TimeServer {

  public static void main(String[] args) {
    DatagramSocket socket = null;
    try {
      socket = new DatagramSocket(3575);
      int n = 1;
      while (n <= 10) {
        byte[] buf = new byte[256];
        // riceve la richiesta
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
```

TimeServer (2)

```
        // produce la risposta
        String dString = new Date().toString();
        buf = dString.getBytes();
        // invia la risposta al client
        InetAddress address = packet.getAddress();
        int port = packet.getPort();
        packet = new DatagramPacket(buf, buf.length, address, port);
        socket.send(packet);
        n++;
      }
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
      socket.close();
    }
  }
}
```

MulticastSocket

Un Multicast Datagram socket è utilizzato per spedire/ricevere messaggi a/da destinatari/mittenti multipli.

Un MulticastSocket è un UDP DatagramSocket con funzionalità aggiuntive per aggregarsi e abbandonare “gruppi” di host.

Un gruppo multicast è identificato da un indirizzo IP di **classe D**. Gli indirizzi di classe D appartengono al range 224.0.0.0 – 239.255.255.255 inclusi.

Tutti i messaggi spediti ad un gruppo multicast verranno ricevuti da tutti i partecipanti al gruppo. Il mittente non deve essere necessariamente un appartenente al gruppo. Le operazioni di adesione e abbandono a/di un gruppo sono regolate dal protocollo **igmp**.

È possibile prenotare permanentemente indirizzi di gruppo multicast o assegnarli e utilizzarli temporaneamente quando necessario sulla propria rete. Per prenotare un indirizzo IP di gruppo permanente per l'utilizzo su Internet, è necessario registrarlo con Internet Assigned Numbers Authority (IANA).
http://www.iana.org/assignments/multicast-addresses

java.net.MulticastSocket

- **MulticastSocket (int port)**
  Crea un MulticastSocket e lo collega alla porta specificata sulla macchina locale.

- **void joinGroup (InetAddress mcastaddr)**
  Si aggiunge ad un multicast group.

- **void leaveGroup (InetAddress mcastaddr)**
  Abbandona un multicast group.

- **void receive (DatagramPacket p)**
  Riceve un DatagramPacket da questo socket.

- **void send (DatagramPacket p)**
  Invia un DatagramPacket su questo socket.

- **void close ()**
  Chiude il MulticastSocket.

MulticastTimeServer (1)

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

public class MulticastTimeServer {

  public static void main(String[] args) {
    MulticastSocket socket = null;
    try {
      socket = new MulticastSocket(3575);
      int n = 1;
      while (n <= 100) {
        byte[] buf = new byte[256];
        // non aspetta la richiesta
        String dString = new Date().toString();
        buf = dString.getBytes();
```

MulticastTimeServer (2)

```
    // invia il messaggio in broadcast
    InetAddress group = InetAddress.getByName("230.0.0.1");
    DatagramPacket packet =new DatagramPacket(buf, buf.length, group, 3575);
    socket.send(packet);
    System.out.println ("Broadcasting: "+dString);
    Thread.sleep(1000);
     n++;
     }
     socket.close();
   }
   catch (Exception e) {
   e.printStackTrace();
   socket.close();
   }
  }
}
```

MulticastTimeClient

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

public class MulticastTimeClient {

  public static void main(String[] args) throws IOException {
    MulticastSocket socket = new MulticastSocket(3575);
    InetAddress group = InetAddress.getByName("230.0.0.1");
    socket.joinGroup(group);
    DatagramPacket packet;
    for (int i = 0; i < 100; i++) {
      byte[] buf = new byte[256];
      packet = new DatagramPacket(buf, buf.length);
      socket.receive(packet);
      String received = new String(packet.getData());
      System.out.println("Time: " + received);
    }
    socket.leaveGroup(group);
    socket.close();
  }
}
```