Applicazioni web

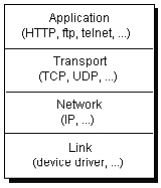
Parte 10

Socket

Alberto Ferrari

Protocolli

- I computer collegati ad Internet comunicano tra loro usando il Transmission Control Protocol (TCP) o lo User Datagram Protocol (UDP)
- Quando si scrivono programmi Java che comunicano
  sulla rete, si programma a livello di *applicazione*
  - Si usano le classi del package *java.net che p*ermettono di comunicare sulla rete In maniera indipendente dalla piattaforma
- E' possibile operare utilizzando il protocollo TCP o il protocollo UDP



Alberto Ferrari

TCP

- TCP (Transmission Control Protocol) è un protocollo basato sulla connessione che garantisce un flusso di dati affidabile tra due computer
- Quando due applicazioni vogliono comunicare tra di loro in maniera affidabile…
  1. Stabiliscono una connessione
  2. Inviano dati nei due sensi su di essa

- ☎ Analogia con una chiamata telefonica
  - TCP garantisce:
    - Che i dati inviati da un capo della connessione arrivino effettivamente all'altro capo
    - Nello stesso ordine in cui sono stati inviati
    - Altrimenti, viene riportato un errore

Alberto Ferrari

Quando utilizzare TCP

- TCP fornisce un canale punto-a-punto per applicazioni che richiedono comunicazioni affidabili. Ad esempio:
  - Hypertext Transfer Protocol (HTTP)
  - File Transfer Protocol (FTP)
  - Telnet

- L'ordine in cui i dati sono inviati e ricevuti è critico per il funzionamento di queste applicazioni
  - Quando si usa HTTP per leggere dati da una url, questi devono essere ricevuti nell'ordine in cui sono inviati
  - Altrimenti, si ottiene un file html confuso, un file zip corrotto o altre informazioni scorrette

Alberto Ferrari

UDP

- UDP (User Datagram Protocol) è un protocollo che invia pacchetti di dati indipendenti, chiamati *datagram*, da un computer ad un altro senza garanzia di consegna
  - UDP non è basato sulla connessione come TCP
  - Il protocollo UDP permette comunicazioni non garantite tra due applicazioni sulla rete
- ✉ Inviare un datagram è simile ad inviare una lettera tramite il servizio postale
  - L'ordine di consegna non è importante e non è garantito
  - Ogni messaggio è indipendente da tutti gli altri

Alberto Ferrari

Quando utilizzare UDP

- La gestione della connessione e il controllo dei dati ricevuti può causare overhead che, in alcuni casi, può invalidare completamente il servizio.
- Per alcuni servizi non ha senso rimandare un dato che è stato ricevuto corrotto.
- Es. Servizio orario che invia l'ora attuale ai suoi client, su richiesta
  - Se il client perde un pacchetto, non ha alcun senso inviarlo di nuovo
  - Al secondo tentativo, l'ora sarà sbagliata nell'istante in cui il client riceverà il pacchetto

Alberto Ferrari

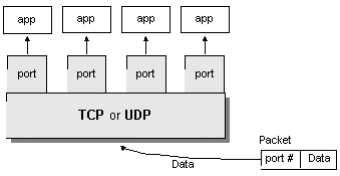
Porte

- In linea di massima, un computer ha una singola connessione fisica alla rete (un solo indirizzo)
  - Tutti i dati destinati ad un particolare computer arrivano attraverso la stessa connessione
  - Tuttavia, i dati potrebbero essere indirizzati a applicazioni diverse in esecuzione sullo stesso computer
- Come fa il computer a sapere a quale applicazione inviare i dati?
- Tramite l'uso delle *porte*
- Tramite diverse porte è possibile che uno stesso computer fornisca diversi servizi (mail, web, telnet, ecc.)

Alberto Ferrari

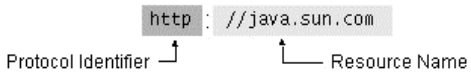
Porte

- I protocolli TCP e UDP usano le porte per mappare i dati in arrivo ad un particolare processo in esecuzione su un computer
  - I dati trasmessi su Internet sono accompagnati da informazione di indirizzo che identifica il computer e la porta cui sono destinati
  - Il computer è identificato dal suo indirizzo IP a 32 bit
  - Le porte sono identificate da un numero a 16 bit, che TCP e UDP usano per consegnare i dati alla giusta applicazione



Alberto Ferrari

URL



- URL è l'acronimo di *Uniform Resource Locator* ed è un riferimento (un indirizzo) per una risorsa su Internet
- Il nome della risorsa è l'indirizzo completo per la risorsa e dipende interamente dal protocollo usato. In HTTP include:
  - *Nome dell'host* – Nome (o indirizzo) della macchina su cui risiede la risorsa
  - *Numero di porta* – Numero di porta a cui connettersi (tipicamente opzionale)
  - *Nome di file* – Percorso (virtuale) del file sulla macchina
  - *Frammento* – Riferimento ad un punto all'interno della risorsa, di solito una locazione all'interno di un file (di solito opzionale)

Alberto Ferrari

Creare URL

- In un programma Java, si può usare una stringa per istanziare un oggetto *URL*

```
URL gamelan = new URL("http://www.gamelan.com/");
```

- L'oggetto URL creato sopra rappresenta una URL assoluta
  - Una URL assoluta contiene tutte le informazioni necessarie per raggiungere la risorsa in questione

Alberto Ferrari

Altri costruttori di URL

- Si possono creare URL partendo da indirizzi relativi
  - Un indirizzo relativo contiene informazioni sufficienti a raggiungere la risorsa solo nel contesto di una URL di base
  - `http://www.gamelan.com/pages/Gamelan.game.html`
    `http://www.gamelan.com/pages/Gamelan.net.html`
  - ```
    URL gamelan = new
    URL("http://www.gamelan.com/pages/");
    URL games = new URL(gamelan, "Gamelan.game.html");
    URL network = new URL(gamelan, "Gamelan.net.html");
    ```
- Costruttori di URL più generici
  - `URL(URL baseURL, String relativeURL)`
  - ```
    URL gamelan = new URL("http", "www.gamelan.com", 80,
    "pages/Gamelan.network.html");
    ```

Alberto Ferrari

MalformedURLException

- Tutti i costruttori di URL generano una eccezione *MalformedURLException* se gli argomenti del costruttore fanno riferimento ad un oggetto nullo o sconosciuto
  - ```
    try {
      URL myURL = new URL(...)
    }
    catch (MalformedURLException e) {
      ...
      // exception handler code here
      ...
    }
    ```

Alberto Ferrari
    ```

TCP - Cos'è una socket

- Una *socket* è un punto terminale di una connessione a doppio senso tra due programmi collegati alla rete
- Una socket è associata ad un numero di porta in modo tale che il livello TCP possa identificare l'applicazione cui i dati sono destinati

Alberto Ferrari

Richiesta di connessione

- Il server aspetta, in ascolto sulla socket, fino all'arrivo di una richiesta di connessione da un client
- Il client conosce il nome della macchina su cui il server gira ed il numero di porta cui è associato
- Per fare una richiesta di connessione, il client prova a contattare il server specificando l'indirizzo e la porta

Alberto Ferrari

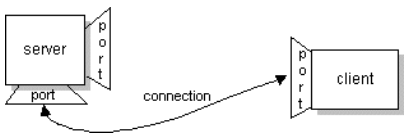
Connessione accettata

- Se tutto va bene, il server accetta la connessione
- All' accettazione, il server ottiene una nuova socket associata ad un nuovo numero di porta
- … in modo da poter continuare ad ascoltare sulla socket originale per altre richieste di connessione mentre soddisfa le richieste del client connesso

Alberto Ferrari

Connessione accettata

- Dal lato client, se la connessione viene accettata, viene creata una socket per comunicare col server
  - Si noti che la socket sul lato client non è legata al numero di porta usato per contattare il server
  - Piuttosto, al client è assegnato un numero di porta locale alla macchina su cui il client è eseguito
- Il client ed il server possono finalmente comunicare attraverso le rispettive socket



TCP - Classi Java

- Il package *java.net* della piattaforma Java fornisce una classe, *Socket*, che implementa un lato della connessione a due sensi tra un programma Java ed un altro programma sulla rete
  - *Socket* si basa su una implementazione dipendente dalla piattaforma…
  - Ma nasconde all'applicazione i dettagli del particolare sistema
  - Usando la classe *java.net.Socket* al posto di codice nativo, un programma Java può comunicare sulla rete in maniera indipendente dalla piattaforma
- Inoltre, *java.net* include la classe *ServerSocket*, che implementa una socket che un server può usare per mettersi in ascolto e accettare connessioni con i client

Alberto Ferrari

Client di esempio (1)

```
// inizializzazione dell'oggetto Socket
// indirizzo IP e numero di porta in ascolto sul server
// caratteristiche imposte dal TCP
Socket client = new Socket("localhost",9999);
// inizializzazione  del canale di comunicazione con il server
// per inviare dati al server si utilizza un PrintWriter
// associato all'output del socket
PrintWriter out = new PrintWriter(client.getOutputStream(),
true);
// Richiesta di una stringa dallo standard input
Scanner tastiera = new Scanner(System.in);
System.out.print("Inserire una stringa: ");
String inputUtente = tastiera.nextLine();
// invio al server tramite il socket
out.println(inputUtente);
```

Alberto Ferrari

Client di esempio

```
// Per ricevere i dati dal server si utilizza un BufferedReader
// associato all'input del socket
BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
// Ricezione dei dati dal server tramite il socket
String inArrivoDalServer = in.readLine();
System.out.println("Dal server arriva: " + inArrivoDalServer);
// chiusura degli stream di comunicazione e dell'oggetto socket
in.close();
out.close();
client.close();
```

Alberto Ferrari

Server di esempio

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

public class ReverseServer {
  public static void main(String[] args) {
    ServerSocket serverSocket = null;
    boolean listening = true;
    try {
      serverSocket = new ServerSocket(4444);
    } catch (IOException e) {
      System.err.println("Could not listen on port: 4444.");
      System.exit(1);
    }

    while (listening) {
      try {
        Socket socket = serverSocket.accept();
        handleClient(socket);
      } catch (IOException e) {
        System.err.println("Accept failed.");
      }
    }
    try { serverSocket.close(); } catch (IOException e) { /* */
}
  }
  ...
```

Server di esempio (1)

```
//inizializzazione dell'oggetto ServerSocket sulla porta 9999
ServerSocket socketSulServer = new ServerSocket(9999)
while (true) {
   System.out.println(“server in attesa...");
   //e' arrivata una richiesta che viene accettata istanziando una
   //socket per comunicare con quel client
   Socket socketFraServerEClient = socketSulServer.accept();
   System.out.println("server in comunicazione con client ...");
   //canale utilizzato per ricevere informazioni dal client
   BufferedReader in = new BufferedReader(new
   InputStreamReader(socketFraServerEClient.getInputStream()));
   //canale utilizzato per inviare comunicazioni al client
   PrintWriter out = new
   PrintWriter(socketFraServerEClient.getOutputStream(), true);
```

Server di esempio (2)

```
//ricezione dal client
String datiDalClient;
datiDalClient = in.readLine();
//costruzione dati dal inviare al client
StringBuffer datiInviatiAlClient;
datiInviatiAlClient = new StringBuffer(datiDalClient).reverse();
//invio dati al client
out.println(datiInviatiAlClient);
//chiusura canali di comunicazione e socket
in.close();
out.close();
socketFraServerEClient.close();
System.out.println("Chiusa la comunicazione con il client");
```

Alberto Ferrari

UDP - Datagram

- Un *datagram* è un messaggio indipendente e auto-contenuto inviato sulla rete il cui arrivo, istante d'arrivo e contenuto non sono garantiti
- Il package *java.net* contiene classi che aiutano ad usare datagram per inviare e ricevere pacchetti sulla rete
  - *DatagramPacket*
  - *DatagramSocket*
  - *MulticastSocket*
- Una applicazione può inviare e ricevere pacchetti di tipo datagram attraverso una *datagram socket*
- Inoltre, i datagram possono essere inviati a destinatari multipli, tutti in ascolto su una *multicast socket*

Datagram – Invio

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

public class DatagramInvio {
  public static void main(String argv[]) throws Exception {
      DatagramSocket socketPerInvio = new DatagramSocket();
      BufferedReader tastiera = new BufferedReader(new InputStreamReader(System.in));
      InetAddress IP = InetAddress.getByName("localhost");
      while ( true ) {
        System.out.println("Inserisci il testo da inviare (esci per uscire): ");
        String testo = tastiera.readLine();       // testo ricevuto in input
        if ( testo.equals("esci") )
          break;
        else {
          byte[] buffer = testo.getBytes();       // conversione in array di byte
          DatagramPacket pacchetto = new DatagramPacket(buffer, buffer.length, IP,
   9999);
          socketPerInvio.send(pacchetto);
          System.out.println("---Dati inviati---");
        }
      }
      socketPerInvio.close();
    }
}
```

Datagram - Ricezione

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

public class DatagramRicezione {
  public static void main(String argv[]) throws Exception {
    DatagramSocket socketPerRicezione = new DatagramSocket(9999);
    while ( true ) {
      byte[] buffer = new byte[255];
      DatagramPacket pacchetto = new DatagramPacket(buffer, buffer.length);
      System.out.println("Attendo dati ...");
      socketPerRicezione.receive(pacchetto);
      String messaggio = new String(pacchetto.getData(),
   pacchetto.getOffset(), pacchetto.getLength());
      System.out.println("Dati ricevuti dall'host " +
   pacchetto.getAddress() +
       " porta " + pacchetto.getPort() + " ... : " + messaggio);
    }
  }
}
```

Alberto Ferrari