AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Applicazioni web

Parte 5

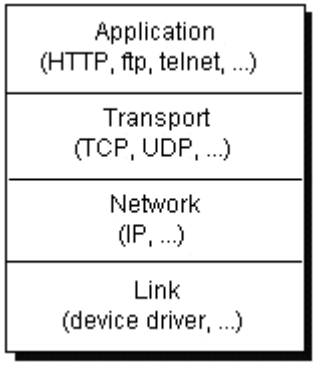
Socket

Michele Tomaiuolo

tomamic@ce.unipr.it

Socket
Protocolli Internet

- 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*
  - Permettono di comunicare sulla rete
  - In maniera indipendente dalla piattaforma



- Non c'è bisogno di preoccuparsi dei dettagli del *trasporto* (TCP o UDP)
  - Per decidere quali classi Java usare in un programma, bisogna conoscere le differenze tra TCP e UDP

Socket 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

☎ Analogo al fare una chiamata telefonica

- Come la compagnia 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

Socket TCP

- TCP fornisce un canale punto-a-punto per applicazioni che richiedono comunicazioni affidabili
  - 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

Socket 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

Socket UDP

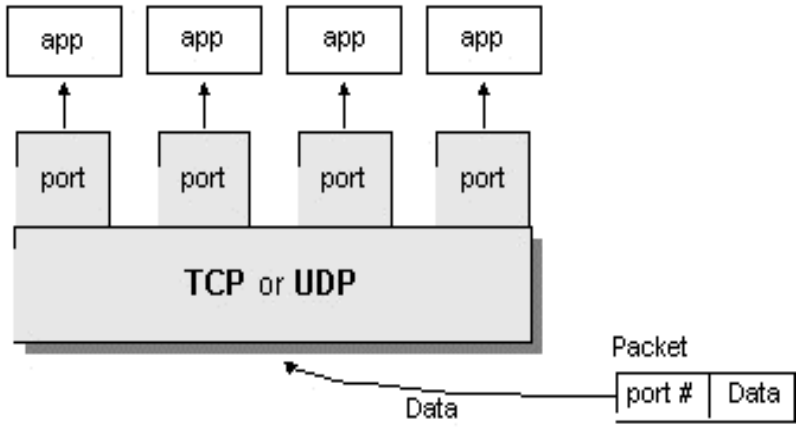
- Qualche applicazione può essere rallentata dall'overhead aggiuntivo
- La connessione affidabile può invalidare completamente il servizio
  - Servizio orario che invia l'ora attuale ai suoi client, su richiesta
  - Se il client perde un pacchetto, non ha alcun senso inviarlo di nuovo
  - Al secondo tentativo, l'ora sarà sbagliata nell'istante in cui il client riceverà il pacchetto

Socket Porte

- In linea di massima, un computer ha una singola connessione fisica alla rete
  - 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*

Socket 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, che IP usa per consegnare i dati al giusto computer sulla rete
  - Le porte sono identificate da un numero a 16 bit, che TCP e UDP usano per consegnare i dati alla giusta applicazione



Socket 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)
  - http://www.ietf.org:80/rfc/rfc2732.txt

Socket
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

Socket
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 game = 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");
    ```

Socket MalformedURLException

- Tutti i costruttori di URL generano una eccezione *MalformedURLException* se gli argomenti del costruttore fanno riferimento ad un protocollo nullo o sconosciuto

- Bisogna catturare e gestire questa eccezione racchiudendo il costruttore in un blocco try/catch

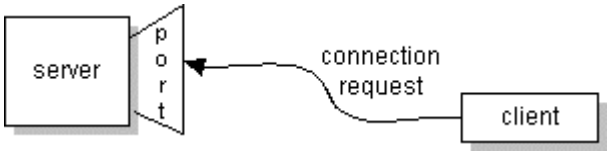
```
try {
   URL myURL = new URL(...)
}
catch (MalformedURLException e) {
   ...
   // exception handler code here
   ...
}
```

Socket
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

Socket
Richiesta di connessione

- Normalmente, una applicazione server gira su un computer specifico ed è legata ad uno specifico numero di porta
- 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 sulla sua specifica macchina e porta



Socket
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

Socket
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

Socket
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

Socket
Client di esempio

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

public class ReverseClient {
  public static void main(String[] args) {

    try {
      Socket socket = new Socket("taranis", 4444);
      handleConnection(socket);
    }
    catch (UnknownHostException e) {
      System.err.println("Don't know about host: taranis.");
      System.exit(1);
    }
    catch (IOException e) {
      System.err.println("Couldn't connect to: taranis.");
      System.exit(1);
    }
  }

  ...
```

Socket
Client di esempio

```
...

static void handleConnection(Socket socket) {
  try {
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(socket.getInputStream()));

    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in));
    String userInput;
    while ((userInput = stdIn.readLine()) != null) {
      out.println(userInput);
      System.out.println("reverse: " + in.readLine());
    }
    out.close(); in.close(); socket.close();
  } catch (IOException e) { /* */ }
}
}
```

Socket
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) { /* */ }
  }
  ...
```

Socket
Server di esempio

```
...
  static void handleClient(Socket socket) {
    try {
      PrintWriter out = new PrintWriter(
        socket.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));

      String inputLine;
      while ((inputLine = in.readLine()) != null) {
        out.println(new StringBuffer(inputLine).reverse());
      }

      out.close(); in.close(); socket.close();
    } catch (IOException e) { /* */ }
  }
}
```

Socket Client multipli

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

public class ReverseMultiServer {
  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();
        Thread t = new ReverseServerThread(socket);
        t.start();
      } catch (IOException e) {
        System.err.println("Accept failed.");
      }
    }
    try { serverSocket.close(); } catch (IOException e) { /* */ }
  }
}
```

Socket Client multipli

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

public class ReverseServerThread extends Thread {
  private Socket socket = null;
  public ReverseServerThread(Socket socket) {
    super("ReverseServerThread"); this.socket = socket;
  }

  public void run() {
    ReverseServer.handleClient(socket);
  }
}
```

Socket 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*

Socket Datagram client

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

public class DatagramClient {
  public static void main(String[] args) throws IOException {
    // bind a datagram socket to any port
    DatagramSocket socket = new DatagramSocket();

    // send request
    InetAddress address = InetAddress.getByName(args[0]);
    DatagramPacket pkt1 =
        new DatagramPacket(new byte[0], 0, address, 4445);
    socket.send(pkt1);

    // get response on the same socket (and port)
    DatagramPacket pkt2 =
        new DatagramPacket(new byte[80], 80);
    socket.receive(pkt2);

    // display response
    String received = new String(pkt2.getData());
    System.out.println("Current date is: " + received);
    socket.close();
  }
}
```

Socket Datagram server

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

public class DatagramServer {
  public static void main(String[] args) {
    DatagramSocket socket = new DatagramSocket(4445);
    while (true) {
      try {
        // receive request
        DatagramPacket pkt1 = new DatagramPacket(new byte[0], 0);
        socket.receive(pkt1);

        byte[] serverDate = new Date().toString().getBytes();

        // get client's "address" and "port" from his own packet
        InetAddress clentAddress = pkt1.getAddress();
        int clientPort = pkt1.getPort();
        DatagramPacket pkt2 = new DatagramPacket(serverDate,
            serverDate.length, clentAddress, clientPort);
        socket.send(pkt2);
      } catch (IOException e) { e.printStackTrace(); }
    }
    socket.close();
  }
}
```