Lezione n.4
LPR-A-09
Indirizzi IP e URL

13/10/2009

Vincenzo Gervasi

PROGRAMMAZIONE DI RETE: INTRODUZIONE

Programmazione di rete:

sviluppare applicazioni definite mediante due o più processi in esecuzione su hosts diversi, distribuiti sulla rete. I processi **cooperano** per realizzare una certa funzionalità

- **Cooperazione**: richiede lo scambio di informazioni (comunicazione) tra i processi
- **Comunicazione** = Utilizza **protocolli** (cioè insieme di regole che i partners della comunicazione devono seguire per poter comunicare)
- Alcuni protocolli utilizzati in INTERNET:
  - IP (Internet Protocol)
  - TCP (Transmission Control Protocol) un protocollo connection-oriented
  - UDP (User Datagram Protocol) protocollo connectionless

PROGRAMMAZIONE DI RETE: INTRODUZIONE

Per identificare un processo con cui si vuole comunicare occorre conoscere:

- la rete in cui si trova l'host su cui e' in esecuzione il processo
- l'host all'interno della rete
- il processo in esecuzione sull'host

  - Identificazione della **rete** e dell'**host**
    - definita dal protocollo **IP (Internet Protocol)**
  - Identificazione del **processo**
    - utilizza il concetto di **porta**
  - **Porta**
    - Intero da 0 a 65535

IL PROTOCOLLO IP

Il Protocollo **IP (Internet Protocol)** definisce

- un sistema di indirizzamento per gli hosts
- la definizione della struttura del pacchetto IP
- un insieme di regole per la spedizione/ricezione dei pacchetti

Due versioni del protocollo IP sono attualmente utilizzate in Internet:

- **IPV4 (IP Versione 4)**
- **IPV6 (IP versione 6)**
  - **IPV6** introduce uno spazio di indirizzi di dimensione maggiore rispetto a **IPV4** (i cui indirizzi cominciano a scarseggiare...)
  - Di uso ancora limitato

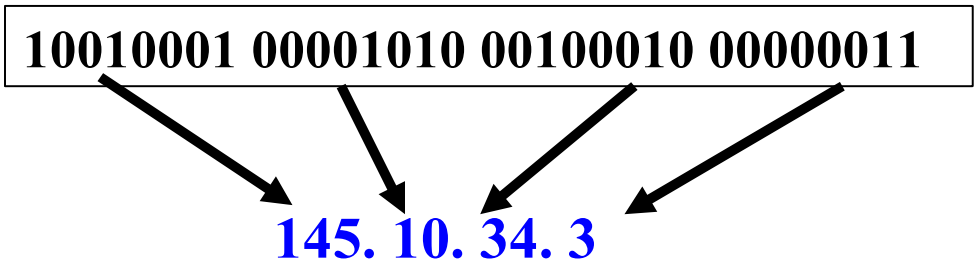
INDIRIZZAMENTO DEGLI HOSTS

Materiale di riferimento: Pitt, capitolo 2

- Ogni host di una rete IPV4 o IPV6 è connesso alla rete mediante **una o più interfacce**
- Ogni interfaccia è caratterizzata da un **indirizzo IP**
- **Indirizzo IP**
  - IPV4: numero rappresentato su 32 bits (4 bytes)
  - IPV6:numero rappresentato su 128 bits (16 bytes)
- **Multi-homed host**: un host che possiede **un insieme di interfacce** verso la rete, e quindi **un insime di indirizzi IP** (uno per ogni interfaccia)
  - **gateway** tra sottoreti IP
  - **routers**

INDIRIZZI IP

Un indirizzo IPV4



- 32 bits
- Ognuno dei 4 bytes, viene interpretato come un numero decimale senza segno
- Rappresentato come sequenza di 4 valori da 0 a 255 separati da "."

Un indirizzo IPV6

- 128 bits
- sequenza di 8 gruppi di 4 cifre esadecimali, separati da ":", es.

2000:fdb8:0000:0000:0001:00ab:853c:39a1

2000:fdb8::1:00ab:853c:39a1

INDIRIZZI IP E NOMI DI DOMINIO

- Gli **indirizzi IP** sono indispensabili per la funzionalità di istradamento dei pacchetti effettuata dai routers, ma sono poco leggibili per gli utenti della rete
- Soluzione: assegnare un **nome simbolico (unico?)** a ogni host
  - si utilizza uno spazio **di nomi gerarchico**, per esempio: **fujih1.cli.di.unipi.it** (host **fujih1** nel dominio **cli.di.unipi.it**)
  - livelli della gerarchia separati dal punto.
  - nomi interpretati da destra a sinistra
- **Risoluzione di indirizzi IP**: corrispondenza tra nomi ed indirizzi IP
- La **risoluzione** viene gestita da un servizio di nomi ("servizio **DNS**")

  DNS = Domain Name System

INDIRIZZAMENTO A LIVELLO DI PROCESSI

- Su ogni host possono essere attivi contemporaneamente più **servizi** (es: e-mail, ftp, http,…)
- Ogni **servizio** viene incapsulato in un diverso **processo**
- L'indirizzamento di un processo avviene mediante una **porta**
- Porta = intero compreso tra 0 e 65535. Non è un dispositivo fisico, ma un'astrazione per individuare i singoli servizi (processi).
- Porte comprese tra 1 e 1023 riservati per particolari servizi.
- In Linux: solo i programmi in esecuzione con diritti di **root** possono ricevere dati da queste porte. Chiunque può inviare dati a queste porte.

  Esempi: porta 7 echo porta 21 ftp
  porta 22 ssh porta 32 telnet
  porta 80 HTTP
- In LINUX: controllare il file **/etc/services**

SOCKETS

- Socket: astrae il concetto di 'communication endpoint'
- Individuato da un indirizzo IP e da un numero di porta
- Socket in JAVA: istanza di una di queste classi (che vedremo in futuro)
  - Socket
  - ServerSocket
  - DatagramSocket
  - MulticastSocket

JAVA: LA CLASSE INETADDRESS

public static InetAddress [ ] *getAllByName (String hostname)*
throws *UnKnownHostException*

utilizzata nel caso di hosts che posseggano piu indirizzi (es: web servers)

public static InetAddress *getLocalHost ()*
throws *UnKnownHostException*

per reperire nome simbolico ed indirizzo IP del computer locale

Getter Methods = Per reperire i campi di un oggetto di tipo InetAddress

public String getHostName ( ) // può fare una reverse resolution

public byte [ ] getAddress ( )

public String getHostAddress ( )

JAVA: LA CLASSE INETADDRESS

public static InetAddress *getByName* (*String* hostname)
throws UnKnownHostException

se il valore di hostname è l'indirizzo IP (una stringa che codifica la dotted form dell'indirizzo IP)

- la getByName *non contatta* il DNS
- il nome dell'host non viene impostato nell'oggetto InetAddress
- il DNS viene contattato solo quando viene richiesto esplicitamente il nome dell'host tramite il metodo getter getHostName( )
- la getHostName( ) non solleva eccezione, se non riesce a risolvere l'indirizzo IP.

JAVA: LA CLASSE INETADDRESS

- accesso al DNS: operazione potenzialmente molto costosa
- i metodi descritti effettuano caching dei nomi/indirizzi risolti.
- nella cache vengono memorizzati anche i tentativi di risoluzione non andati a buon fine (di default: per un certo numero di secondi)
- se creo un InetAddress per lo stesso host, il nome viene risolto con i dati nella cache (di default: per sempre)
- permanenza dati nella cache: per default alcuni secondi se la risoluzione non ha avuto successo, illimitato altrimenti. Problemi: indirizzi dinamici..
- java.security.Security.setProperty consente di impostare il numero di secondi in cui una entry nella cache rimane valida, tramite le proprietà

networkaddress.cache.ttl e networkaddress.cache.negative.ttl

esempio:

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
```

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

- implementazione in JAVA della utility UNIX *nslookup*
- nslookup
  - consente di tradurre nomi di hosts in indirizzi IP e viceversa
  - i valori da tradurre possono essere forniti in modo interattivo oppure da linea di comando
  - si entra in modalità interattiva se non si forniscono parametri da linea di comando
  - consente anche funzioni più complesse (vedere LINUX)

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

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

public class HostLookUp {
    public static void main (String [ ] args) {
      if (args.length > 0) {
       for (int i=0; i<args.length; i++) {
            System.out.println (lookup(args[i])) ;
                        }
      }
      else {/* modalita' interattiva*/ }
```

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

```
private static boolean isHostName (String host)
{
      char[ ] ca = host.toCharArray();
      for (int i = 0; i < ca.length; i++) {
            if(!Character.isDigit(ca[i])) {
                  if (ca[i] != '.')
                        return true;
            }
      }
      return false;
}
```

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

```
private static String lookup(String host) {
      InetAddress node;
      try {
            node = InetAddress.getByName(host);
            System.out.println(node);
            if (isHostName(host))
                  return node.getHostAddress( );
            else
                  return node.getHostName ( );
       } catch (UnknownHostException e)
                  return "non ho trovato l'host";
}
```

Esercizio 1

- Scrivere un programma Java Resolve che traduca una sequenza di nomi simbolici di host nei corrispondenti indirizzi IP.
- Resolve legge i nomi simbolici da un file, il cui nome è passato da linea di comando oppure richiesto all'utente.
- Si deve definire un task che estenda l'interfaccia Callable, e che, ricevuto come parametro un nome simbolico, provvede a tradurre il nome ritornando un InetAddress.
- Per ottimizzare la ricerca, si deve attivare un pool di thread che esegua i task in modo concorrente. Ogni volta che si sottomette al pool di thread un task, si ottiene un oggetto Future<InetAddress>, che deve essere aggiunto ad un ArrayList.
- Infine, si scorre l'ArrayList, stampando a video gli InetAddress.

Esercizio 2

- Scrivere un programma che enumeri e stampi a video tutte le interfacce di rete del computer, usando i metodi della classe java.net.NetworkInterface.
- Usare il metodo statico getNetworkInterfaces() per ottenere una Enumeration di NetworkInterface.
- Per ogni NetworkInterface, stampare gli indirizzi IP associati ad essa (IPv4 e IPv6) e il nome dell'interfaccia.

Esercizio 3

- Scrivere un programma che ricerca una parola chiave (key) nei file contenuti in una directory (fornita dall'utente) e nelle sue sottodirectory. Per ogni file che contiene key, si deve visualizzare il nome dei file e il contenuto della prima riga trovata che contiene key.
- Creare una classe FindKeyword che implementa Callable, alla quale si può passare una directory come parametro del costruttore, e che ritorna un array di stringhe del formato

  <nome file> : <contenuto riga che contiene key>
- Il metodo search della classe FindKeyword implementa la ricerca di key all'interno di un singolo file, e ritorna una stringa formattata come sopra oppure null se key non compare.
- Creare un pool di thread a cui vengono sottomessi un task FindKeyword per ogni directory/sottodirectory, e usare gli oggetti Future restituiti per stampare a video i risultati ottenuti.

Uniform Resource Locator

- **URL** è un acronimo per Uniform Resource Locator
- Un riferimento (un indirizzo) per una **risorsa** su Internet.
  - Di solito un URL è il nome di un file su un host.
  - Ma può anche puntare ad altre risorse:
    - una query per un database;
    - l'output di un comando.
  - Es: http://java.sun.com
    - http: identificativo del protocollo.
    - java.sun.com: nome della risorsa.

Nomi di risorsa

- Il **nome di una risorsa** è composto da:
  - Host Name: il nome dell'host su cui si trova la risorsa.
  - Filename: il pathname del file sull'host.
  - Port Number: il numero della porta su cui connettersi (opzionale).
  - Reference: un riferimento ad una specifica locazione all'interno del file (opzionale).
- Nel caso del protocollo http, se il Filename è omesso (o finisce per /), il web server è configurato per restituire un file di default all'interno del path (ad es. index.html, index.php, index.asp).

URL e URI

- Un **URI** (Uniform Resource Identifier) è un costrutto sintattico che specifica, tramite le varie parti che lo compongono, una **risorsa** su Internet:
  - [schema:]ParteSpecificaSchema[#frammento]
  - dove ParteSpecificaSchema ha la struttura
    - [//autorita'][percorso][?query]
- Un URL è un tipo particolare di URI: contiene sufficienti informazioni per individuare e ottenere una risorsa.
- Altre URI, ad es: URN:ISBN:0-395-36341-1 non specificano come individuare la risorsa.
  - In questo caso, le URI sono dette **URN** (Uniform Resource Name).

URL in Java

- In JAVA per creare un **oggetto URL**:
  - `URL cli = new URL("http://www.cli.di.unipi.it/");`
  - è un esempio di un URL assoluto.
- È anche possibile creare un URL relativo, che ha la forma
  - `URL(URL baseURL, String relativeURL)`
  - Esempi:.
    - `URL cli = new URL("http://www.cli.di.unipi.it/");`
    - `URL faq = new URL(cli, "faq");`
  - che risulterà puntare a http://www.cli.di.unipi.it/faq
- I protocolli gestiti da Java con gli URL sono http, https, ftp, file e jar.
- I costruttori possono lanciare una MalformedURLException.

Parsare un URL

- La classe URL offre metodi per accedere ai componenti di una URL

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

public class URLReader {
    public static void main(String[] args) throws Exception {
        String url = "http://www.cli.di.unipi.it:80/faq";
        URL cli = new URL(url);
        System.out.println("protocol = " + cli.getProtocol());
        System.out.println("authority = " + cli.getAuthority());
        System.out.println("host = " + cli.getHost());
        System.out.println("port = " + cli.getPort());
        System.out.println("path = " + cli.getPath());
        System.out.println("query = " + cli.getQuery());
        System.out.println("filename = " + cli.getFile());
        System.out.println("ref = " + cli.getRef());
    }
}
```

Parsare un URL

- Eseguendo l'esempio precedente si ottiene:

```
protocol = http
authority = www.cli.di.unipi.it:80
host = www.cli.di.unipi.it
port = 80
path = /faq
query = null
filename = /faq
ref = null
```

Leggere da un URL

- Una volta creato un oggetto URL si può invocare il metodo **openStream()** per ottenere uno stream da cui poter leggere il contenuto dell'URL.
- Il metodo openStream() ritorna un oggetto java.io.InputStream
  - leggere da un URL è analogo a leggere da uno stream di input.

```
import java.net.*;
import java.io.*;
public class URLReader {
    public static void main(String[] args) throws Exception {
        URL cli = new URL("http://www.cli.di.unipi.it/");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(cli.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Leggere da un URL

- Una volta creato un oggetto URL si può invocare il metodo **openStream()** per ottenere uno stream da cui poter leggere il contenuto dell'URL.
- Il metodo openStream() ritorna un oggetto java.io.InputStream
  - leggere da un URL è analogo a legge


```
import java.net.*;
import java.io.*;
public class URLReader {
    public static void main(String[] args) throws Exception {
        URL cli = new URL("http://www.cli.di.unipi.it/");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(cli.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Leggere da un URL

- Eseguendo l'esempio precedente, si ottiene:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="it">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<link rel="stylesheet" href="/cdc.css" type="text/css">
<link rel="alternate" type="application/rss+xml"
title="Ultime notizie" href="/feed.php">
<title>Home CdC </title>
</head>
<body bgcolor="#ced8e0">
....
```

- Può essere necessario impostare il proxy su Java:

```
java -Dhttp.proxyHost=proxyhost [-Dhttp.proxyPort=portNumber]
URLReader
```

Connettersi a un URL

- Nell'esempio precedente, la connessione all'URL veniva effettuata solo dopo aver invocato openStream().
- In alternativa, è possibile invocare il metodo **openConnection()** per ottenere un oggetto **URLConnection**.
  - Utile nel caso in cui si vogliano settare alcuni parametri o proprietà della richiesta prima di connettersi.
    - **es:** `cliConn.setRequestProperty("User-Agent", "Mozilla/5.0");`
- Successivamente, si invoca **URLConnection.connect()**.

```
URL cli = new URL("http://www.cli.di.unipi.it/");
URLConnection cliConn = cli.openConnection();
cliConn.connect();
BufferedReader in = new BufferedReader(
    new InputStreamReader(cliConn.getInputStream()));
```

URL e HTTPS

- Tutto quanto detto vale anche per le connessioni sicure via HTTPS

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

public class SecureClientUrl {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://www.verisign.com");
            URLConnection conn = url.openConnection();
            BufferedReader in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            in.close();
        } catch (Exception e){e.printStackTrace();}
    }
}
```