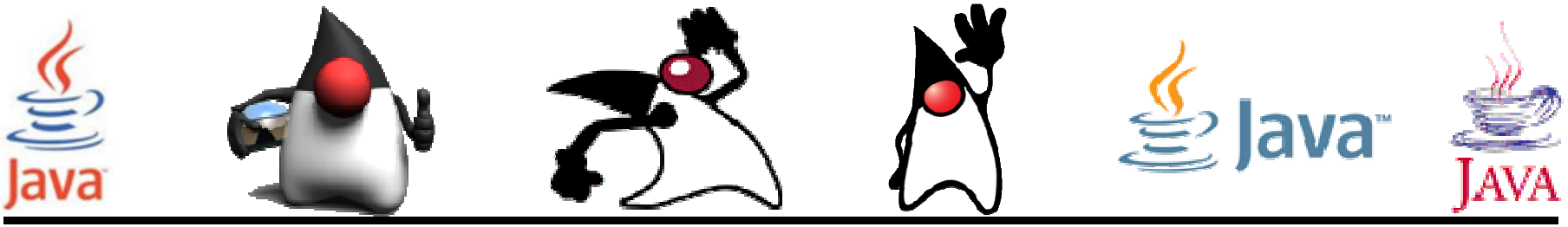


Networking

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Aprender los conceptos básicos de redes y su implementación en Java
- Desarrollar programas de comunicación

Índice

- Conceptos de redes
- Basic Concepts on Networking
  - Dirección IP
  - Protocolo
  - Puertos
  - Paradigma Cliente/Servidor
  - Sockets
- Paquete Java Networking
  - La clase ServerSocket y Socket
  - La clase MulticastSocket y DatagramPacket

Concepto básico de redes – dirección IP

- Similar, desde el punto de vista lógico, al tradicional correo tradicional
  - Cada dirección identifica de manera única un objeto particular
- Cada computador conectado a Internet tiene una dirección IP (Internet Protocol)
- Se usa un número de 32 bits para identificar de manera única cada computador conectado a Internet
  - 193.144.189.63
  - macc.unican.es

Protocolo

- Los protocolos son necesarios porque hay diferentes tipos de comunicación sobre Internet
- Cada tipo de comunicación requiere un protocolo específico y único
- Un protocolo se puede definir como el conjunto de reglas y estándares que definen un cierto tipo de comunicación Internet
- El protocolo describe el formato de dato enviado a Internet (cómo y cuando es enviado)
  - Los protocolos los usamos habitualmente (protocolo social)

Protocolo

- Algunos protocolos importantes usados en Internet son:
  - File Transfer Protocol (FTP), que permite transferir ficheros sobre Internet
  - Hypertext Transfer Protocol (HTTP), usada para transferir doumentos HTML sobre la Web
- Ambos protocolos tienen su propio conjunto de reglas y estándares en la forma en que los datos se transfieren
- Java proporciona el soporte para ambos protocolos

Puertos

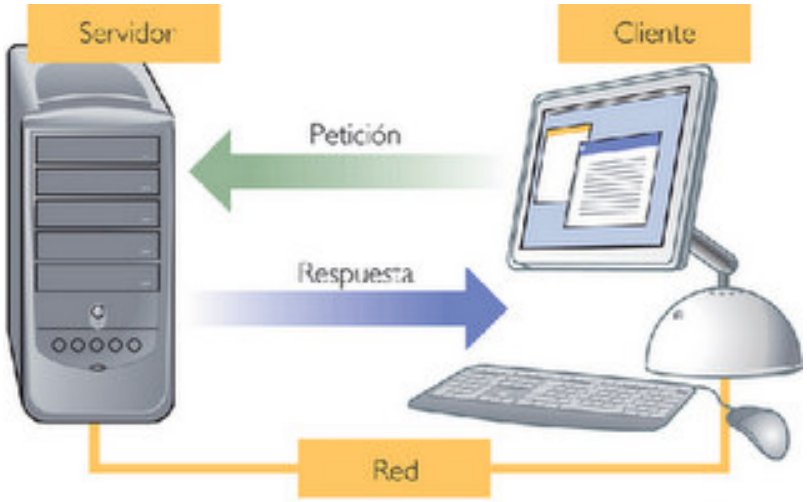
- Los protocolos tienen sentido cuando se usan en el contexto de un servicio
  - El protocolo HTTP es usado cuando se proporciona contenido Web a través del servicio HTTP
  - Cada computador en Internet puede proporcionar una variedad de servicios
- Para establecer una línea de comunicación a través de un protocolo y usar un servicio particular específico requiere la conexión al puerto apropiado

Puertos

- Un puerto es un número de 16 bits que identifica cada servicio ofrecido por el servidor de red
- Puertos estándar
  - Números asociados a un tipo de servicio particular. Ejemplos: el puerto del servicio FTP es 21, el puerto del servicio HTTP es 80, el puerto del servicio TELNET es 23
  - Los valores de puerto son < 1024
- Valores de puertos >= 1024
  - Disponibles para una comunicación personalizada
  - Si ya está en uso, debe buscarse otros valores no usados

El paradigma cliente/servidor

- Básico en la plataforma de red Java
- Involucra dos elementos principales:
  - Cliente, máquina que necesita de algún tipo de información
  - Servidor, máquina que almacena información y que espera por responder a una petición



Escenario:

- El cliente se conecta al servidor y solicita cierta información
- El servidor recibe la petición y responde al cliente con la información

Sockets - definiciones

- Abstracción software para un medio de comunicación de entrada o salida
- Canales de comunicación que permiten transferir datos a través de un puerto en particular
- Punto de comunicación o de conexión lógica entre dos máquinas
- Tipo particular de comunicación en red usada por Java en la programación de redes
- Java realiza toda la comunicación de bajo nivel a través de sockets

Sockets - tipos

- Hay dos tipos de sockets:
  - Servidores
  - Clientes
- Un socket servidor espera las solicitudes de conexión de clientes.
- Un socket cliente se puede usar para enviar y recibir datos.

Package de Networking Java

- Java proporciona el paquete java.net que proporciona clases útiles para el desarrollo de aplicaciones de red
- Algunas clases en el package son:
  - ServerSocket
  - Socket
  - MulticastSocket
  - DatagramSocket

La clase ServerSocket

- Proporciona la funcionalidad básica de un servidor
- Escuchan un *puerto* específico, consistente en un número único
  - El número del puerto es necesario para distinguir diferentes servidores ejecutándose en una misma máquina
- Deben ejecutarse antes que los clientes inicien la comunicación

La clase ServerSocket

Resumen de Constructores

ServerSocket()
Creates an unbound server socket.

ServerSocket(int port)
Creates a server socket, bound to the specified port.

ServerSocket(int port, int backlog)
Creates a server socket and binds it to the specified local port number, with the specified backlog.

ServerSocket(int port, int backlog, InetAddress bindAddr)
Create a server with the specified port, listen backlog, and local IP address to bind to.

La clase ServerSocket - creación

- En su forma más simple se crean instanciando la clase ServerSocket mediante cualquiera de los siguientes constructores:

```
ServerSocket(int port)
ServerSocket(int port, int backlog)
```

- *port* es el número de puerto que atiende el socket.
- Cuando varios clientes contactan con el servidor al mismo tiempo, se colocan en una cola y se atienden en el orden de recepción.

La clase ServerSocket - métodos

- Métodos usados comúnmente:

  accept() Espera por una conexión. El thread que ejecuta el método se bloqueará hasta que se recibe una solicitud, devolviendo un socket cliente.

  close() para recibir solicitudes de clientes.

- La comunicación con sockets se gestiona con input / output streams.

Ejemplo de ServerSocket

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

public class NetworkingServer {
    public static void main(String [] args) {
        ServerSocket server = null;
        Socket client;
        // Numero de puerto a usar por defecto
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }
        // Crea socket lado Server
        try {
            server = new ServerSocket(portnumber);
        } catch (IOException ie) {
            System.out.println("No puede abrir socket." + ie);
            System.exit(1);
        }
        System.out.println("ServerSocket creado" + server);
```

Ejemplo de ServerSocket

```
while(true) {// Espera datos del cliente y responde
    try {
        // Escucha por una conexión al socket y lo acepta.
        System.out.println("Esperando solicitud de conexion...");
        client = server.accept();
        System.out.println("Conexion aceptada...");
        String clientHost =
                client.getInetAddress().getHostAddress();
        int clientPort = client.getPort();
        System.out.println("Cliente = " + clientHost + " Puerto
                Cliente = " + clientPort);
        // Lee datos del cliente
        InputStream clientIn = client.getInputStream();
        BufferedReader br = new BufferedReader(new
                        InputStreamReader(clientIn));
        String msgFromClient = br.readLine();
        System.out.println("Mensaje recibido del cliente = " +
                        msgFromClient);
```

Ejemplo de ServerSocket

```
                // Envia respuesta al cliente
                if (msgFromClient != null &&
                        !msgFromClient.equalsIgnoreCase("adios")) {
                    OutputStream clientOut = client.getOutputStream();
                    PrintWriter pw = new PrintWriter(clientOut, true);
                    String ansMsg = "Hola, " + msgFromClient;
                    pw.println(ansMsg);
                }
                // Cierra sockets
                if (msgFromClient != null &&
                        msgFromClient.equalsIgnoreCase("adios")) {
                    server.close();
                    client.close();
                    break;
                }
            } catch (IOException ie) {
          }
        }
    }
}
```

Ejemplo de ServerSocket

```
                // Envia respuesta al cliente
                if (msgFromClient != null &&
                        !msgFromClient.equalsIgnoreCase("adios")) {
                    OutputStream clientOut = client.getOutputStream();
                    PrintWriter pw = new PrintWriter(clientOut, true);
                    String ansMsg = "Hola, " + msgFromClient;
                    pw.println(ansMsg);
                }
                // Cierra sockets
                if (msgFromClient != null &&
                        msgFromClient.equalsIgnoreCase("adios")) {
                    server.close();
                    client.close();
                    break;
                }
            } catch (IOException ie) {
          }
        }
    }
}
```

La clase Socket (Cliente)

- Implementa un cliente socket. Se consigue de dos formas:

  `lado cliente: Socket(String host, int port)`
  `lado servidor: método accept() de ServerSocket`

- Cada cliente socket tiene un objeto InputStream para recibir datos y un objeto OutputStream para enviar datos.

- Métodos usados:

  `getInputStream()` Devuelve un objeto InputStream
  `getOutputStream()` Devuelve un objeto OutputStream
  `close()` Cerrar la conexión

La clase Socket (Cliente)

Resumen de Constructores

Socket()
Creates an unconnected socket, with the system-default type of SocketImpl.

Socket(InetAddress address, int port)
Creates a stream socket and connects it to the specified port number at the specified IP address.

Socket(InetAddress address, int port, InetAddress localAddr, int localPort)
Creates a socket and connects it to the specified remote address on the specified remote port.

Socket(Proxy proxy)
Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.

protected **Socket**(SocketImpl impl)
Creates an unconnected Socket with a user-specified SocketImpl.

Socket(String host, int port)
Creates a stream socket and connects it to the specified port number on the named host.

Socket(String host, int port, InetAddress localAddr, int localPort)
Creates a socket and connects it to the specified remote host on the specified remote port.

Ejemplo de cliente Socket

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

public class NetworkingClient {
    public static void main(String args[]) {
        Socket client = null;
        String host; // Host y Puerto a considerar por defecto
        int portnumber;
        if (args.length >= 1){
            host = args[0];
            portnumber = Integer.parseInt(args[1]);
        } else { host = "localhost";   portnumber = 1234; }
        for (int i=0; i <10; i++) {
            try {
                String msg = "";
                // Creacion cliente socket
                client = new Socket(host, portnumber);
                System.out.println("Cliente socket creado " + client);
                // Creacion de un output stream del cliente socket
                OutputStream clientOut = client.getOutputStream();
                PrintWriter pw = new PrintWriter(clientOut, true);
```

Ejemplo de cliente Socket

```
            // Creacion de un input stream del cliente socket
            InputStream clientIn = client.getInputStream();
            BufferedReader br = new BufferedReader(new
                    InputStreamReader(clientIn));
            // Creacion de BufferedReader para el standard input
            BufferedReader stdIn = new BufferedReader(new
                    InputStreamReader(System.in));
            System.out.println("Escribir nombre. Para salir adios);
            // Lee datos del standard input y lo escribe
            // en el output stream del cliente socket.
            msg = stdIn.readLine().trim(); pw.println(msg);
            // Lee datos del input stream del cliente socket.
            System.out.println("Mensaje recibido del seridor = " +
                                br.readLine());
            pw.close(); br.close(); client.close(); // cierra Streams
            if (msg.equalsIgnoreCase("adios")) {// Para la operacion
                break;
            }
        } catch (IOException ie) {
            System.out.println("Error I/O " + ie);
        } } } }
```

La clase DatagramPacket

- Usada para entregar datos con protocolo sin conexión
- El problema es que no está garantizado la entrega de paquetes

Resumen de Constructores

DatagramPacket(byte[] buf, int length)
Constructs a DatagramPacket for receiving packets of length length.

DatagramPacket(byte[] buf, int length, InetAddress address, int port)
Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

DatagramPacket(byte[] buf, int offset, int length)
Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer.

DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)
Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)
Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

DatagramPacket(byte[] buf, int length, SocketAddress address)
Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

La clase MulticastSocket

- Util para aplicaciones que implementan comunicaciones en grupo sobre datagramas UDP
- La dirección IP para un grupo multicast se encuentra en el rango 224.0.0.0 (no usar!) a 239.255.255.255

Resumen de Constructores

MulticastSocket()
Create a multicast socket.

MulticastSocket(int port)
Create a multicast socket and bind it to a specific port.

MulticastSocket(SocketAddress bindaddr)
Create a MulticastSocket bound to the specified socket address.

La clase MulticastSocket

- Envío de mensaje a un grupo
  - Debe ser un miembro del grupo multicast mediante el uso del método `joinGroup`
  - Usar el método `send`
  - Usar el método `leaveGroup`
- El método `send` requiere como argumento un objeto `DatagramPacket`

Ejemplo de Server MulticastSocket

```
import java.net.*;

public class MulticastChatServer {
    public static void main(String args[]) throws Exception {
        // Numero de puerto a usar por defecto
        int portnumber = 5000;
        if (args.length >= 1) {
            portnumber = Integer.parseInt(args[0]);
        }
        // Creacion de MulticastSocket
        MulticastSocket serverMulticastSocket =
                new MulticastSocket(portnumber);
        System.out.println("MulticastSocket creado e puerto " +
                           portnumber);
        // Determina la direccion IP delhost, dado el host name
        InetAddress group = InetAddress.getByName("225.4.5.6");
        // getByName- returna direccion IP de un host dado
        serverMulticastSocket.joinGroup(group);
        System.out.println("metodo joinGroup invocado...");
```

Ejemplo de Server MulticastSocket

```
        boolean infinite = true;
        // Continuamente recibe datos y los imprime
        while (infinite) {
            byte buf[] = new byte[1024];
            DatagramPacket data = new DatagramPacket(buf, buf.length);
            serverMulticastSocket.receive(data);
            String msg = new String(data.getData()).trim();
            System.out.println("Mensaje recibido por el cliente = " +
                               msg);
        }
        serverMulticastSocket.close();
    }
}
```

Ejemplo de Cliente MulticastSocket

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

public class MulticastChatClient {
    public static void main(String args[])
        throws Exception {
        // Numero de puerto a usar por defecto
        int portnumber = 5000;
        if (args.length >= 1) {
            portnumber = Integer.parseInt(args[0]);
        }
        // Creacion de un MulticastSocket
        MulticastSocket chatMulticastSocket = new
                                        MulticastSocket(portnumber);
        // Determina la direccion IP de un host, dado el host name
        InetAddress group = InetAddress.getByName("225.4.5.6");
        // Se une al multicast group
        chatMulticastSocket.joinGroup(group);
```

Ejemplo de Cliente MulticastSocket

```
        // Pide al usuario ingresar un mensaje
        String msg = "";
        System.out.println("Escribir mensaje al servidor: ");
        BufferedReader br =
                new BufferedReader(new InputStreamReader(System.in));
        msg = br.readLine();
        // Envio del mensaje a la direccion Multicast
        DatagramPacket data = new DatagramPacket(msg.getBytes(), 0,
                msg.length(), group, portnumber);
        chatMulticastSocket.send(data);
        // Cierre del socket
        chatMulticastSocket.close();
    }
}
```

Computación distribuída

- Mecanismos de Java para soportar la computación distribuída:
  - Comunicación basada en socket
  - RMI (remote method invocation)
- Mecanismos de interfaz entre aplicaciones Java y no-Java:
  - JDBC (Java DataBase Connectivity)
  - CORBA (Common Object Request Broker Architecture).