Sockets datagram

Aunque el principal objetivo es la utilización de los *sockets stream* para diseñar aplicaciones distribuidas cliente-servidor, vamos a ver también las características y un pequeño ejemplo del uso de los *sockets datagram*.

1. No están orientados a conexión, es decir, no se abre una comunicación y por lo tanto tampoco se cierra.

2. El proceso se parece bastante al envío de una carta postal, escribo una dirección y envío un mensaje a dicha dirección, una vez hecho me despreocupo de lo que ocurra en el destino.

3. Utilizan el protocolo UDP en la capa de transporte dentro de la pila de protocolos IP.

4. No se puede garantizar al 100% que los mensajes lleguen al destino y tampoco que lleguen en el orden correcto.

Siguiendo la filosofía del CLIENTE/SERVIDOR de sockets streams

El programa que vamos a ver se dedica exclusivamente a enviar mensajes, aunque un sólo programa podría

enviar y recibir. Enviará los mensajes que escriba el usuario por teclado a la dirección IP =192.168.1.104 y al puerto 5000.

El programa terminará cuando el usuario introduzca el texto FIN.

Clase DatagramSocket

El constructor de la clase DatagramSocket permite crear un socket UDP para enviar y recibir paquetes.

- **DatagramSocket()**: Crea un socket sin puerto específico.
- **DatagramSocket(int port)**: Crea un socket asociado a un puerto específico.
- **DatagramSocket(int port, InetAddress bindAddr)**: Crea un socket asociado a un puerto y a una dirección IP específicos.

Los métodos principales de la clase DatagramSocket son:

send(DatagramPacket p): Envía un paquete de datos (un DatagramPacket) a través del socket.
receive(DatagramPacket p): Recibe un paquete de datos en el socket y lo almacena en el DatagramPacket proporcionado.
getLocalAddress(): Devuelve la dirección IP local de la máquina que está utilizando el socket.
getLocalPort(): Devuelve el puerto local del socket.
close(): Cierra el socket, liberando los recursos asociados.
setSoTimeout(int timeout): Establece un tiempo de espera (timeout) para operaciones de recepción, en milisegundos.
getSoTimeout(): Obtiene el tiempo de espera configurado para las operaciones de recepción.
bind(SocketAddress address): Asocia el socket a una dirección y puerto específicos.
isClosed(): Verifica si el socket está cerrado.

Clase DatagramPacket

Es una clase en Java utilizada para representar un paquete de datos en el protocolo UDP. Se utiliza para enviar o recibir datos a través de sockets UDP.

- **Constructor**: DatagramPacket(byte[] buf, int length, InetAddress address, int port)
  - **buf**: El arreglo de bytes que contiene los datos que se enviarán o recibirán.
  - **length**: El número de bytes que se deben enviar o recibir del arreglo buf.
  - **address**: La dirección IP o el host al que se enviará el paquete.
  - **port**: El puerto en el que el paquete será recibido o enviado.

Los métodos principales de la clase DatagramPacket son:

getData(): Obtiene los datos (en bytes) del paquete.
getLength(): Devuelve la longitud de los datos en el paquete.
getOffset(): Indica el índice de inicio de los datos dentro del arreglo de bytes.
getAddress(): Devuelve la dirección IP (o nombre de host) del destino.
getPort(): Devuelve el puerto de destino al que se envió el paquete.
setData(): Establece los datos (bytes) del paquete.
setLength(): Establece la longitud de los datos.
setOffset(): Establece el índice desde donde empiezan los datos en el arreglo de bytes.
setAddress(): Establece la dirección IP o nombre del host del destino.
setPort(): Establece el puerto de destino.

El código envía un mensaje a través de UDP usando DatagramPacket.

DatagramPacket carta = new DatagramPacket(mensaje.getBytes(), lon, destino, 5000);

mensaje.getBytes(): Convierte el mensaje (una cadena de texto) en un arreglo de bytes.
lon: Es la longitud del arreglo de bytes que se enviará.
Representa el número de bytes que se deben leer del arreglo mensaje.getBytes().
En otras palabras, lon limita el tamaño de los datos que se envían.
destino: Es la dirección IP o el nombre del host al que se enviará el paquete.
5000: Es el puerto de destino al que se enviará el paquete.

ds.send(carta);: Envia el paquete carta a través del socket ds.

Opciones para lon

La opción que puedes poner en lon depende de cuántos bytes del mensaje quieres enviar. Si el mensaje es largo y solo necesitas enviar una parte de él, pones un valor menor que el total del arreglo de bytes. Si deseas enviar todo el mensaje, lon debería ser igual a mensaje.getBytes().length.

```
String mensaje = "Hola, esto es un mensaje";
byte[] bytesMensaje = mensaje.getBytes();
int lon = bytesMensaje.length; // lon será 24 en este caso, que es la longitud del mensaje
DatagramPacket carta = new DatagramPacket(bytesMensaje, lon, destino, 5000);
ds.send(carta);
```

```
String mensaje = "Hola, esto es un mensaje";
byte[] bytesMensaje = mensaje.getBytes();
int lon = 5; // Solo los primeros 5 bytes del mensaje ("Hola ")
DatagramPacket carta = new DatagramPacket(bytesMensaje, lon, destino, 5000);
ds.send(carta);
```

```
String mensaje = "Este es un mensaje muy largo que será truncado";
byte[] bytesMensaje = mensaje.getBytes();
int lon = Math.min(bytesMensaje.length, 50); // Enviar hasta 50 bytes, si el mensaje es más largo
DatagramPacket carta = new DatagramPacket(bytesMensaje, lon, destino, 5000);
ds.send(carta);
```

DatagramSocket ds = new DatagramSocket();

Creamos el socket datagram como un objeto de la clase *DatagramSocket*.

InetAddress destino = InetAddress.getByName("192.168.1.104");

La clase *InetAddress* representa una dirección IP.

DatagramPacket carta = new DatagramPacket(mensaje.getBytes(), lon, destino, 5000);

ds.send(carta);

Podríamos decir que esto equivale a enviar una carta. La clase *DatagramPacket* representa un paquete a enviar. Construimos el objeto *carta* con los siguientes argumentos: el mensaje a enviar, la longitud del mensaje, el destino (objeto *InetAddress*), el puerto).

Con el método *send()* de la clase *DatagramSocket* enviamos el paquete (objeto *DatagramPacket*) especificado en el argumento.