

Sockets y Threads en JAVA

Luis Mera Castellón e6795058@est.fib.upc.es
Jordi Romero Rojo e6705701@est.fib.upc.es

A spiral-bound notebook with a brown cover and a white page. The spiral binding is on the left side. The word "Sockets" is written in a large, black, serif font in the center of the page.

Sockets

Qué es un socket?

- Definición: Un socket es un punto final de un enlace de comunicación de dos vías entre dos programas que se ejecutan a través de la red.
- El cliente y el servidor deben ponerse de acuerdo sobre el protocolo que utilizarán.

Tipos de socket

- Hay dos tipos de socket:
 - Orientado a conexión:
 - Establece un camino virtual entre servidor y cliente, fiable, sin pérdidas de información ni duplicados, la información llega en el mismo orden que se envía.
 - El cliente abre una sesión en el servidor y este guarda un estado del cliente.
 - El cliente utiliza la clase Socket
 - El servidor utiliza la clase ServerSocket

Tipos de socket

– No orientado a conexión:

- Envío de datagramas de tamaño fijo. No es fiable, puede haber pérdidas de información y duplicados, y la información puede llegar en distinto orden del que se envía.
- No se guarda ningún estado del cliente en el servidor, por ello, es más tolerante a fallos del sistema.

– Tanto el cliente como el servidor utilizan la clase DatagramSocket.

- Todas estas clases (Socket, ServerSocket y DatagramSocket) se encuentran en el paquete java.net

Clase Socket

- Constructores:

- `public Socket ()`
- `public Socket (InetAddress address, int port)`
- `public Socket (String host, int port)`
- `public Socket (InetAddress address, int port, InetAddress localAddr, int localPort)`
- `public Socket (String host, int port, InetAddress localAddr, int localPort)`

- `address / localAddr`: dirección IP de la máquina remota / local.
- `port / localPort`: puerto de la máquina remota / local.
- `host`: nombre de la máquina remota

- En el caso del primer constructor crea un socket sin conexión.

Servicios de la clase Socket

- `public InetAddress getInetAddress():`
 - Devuelve la dirección IP de la máquina en la que estamos conectados.
- `public int getPort():`
 - Devuelve el puerto de la máquina remota.
- `public void close()`
 - Cierra el canal de comunicación.
- `public InputStream getInputStream():`
 - Devuelve el canal de lectura del socket.
- `public OutputStream getOutputStream():`
 - Devuelve el canal de escritura del socket.

- Además JAVA proporciona dos llamadas para saber la @IP y puerto local (`getLocalAddress()` y `getLocalPort()`)

Clase ServerSocket

- Constructores:

- `public ServerSocket (int port)`
- `public ServerSocket (int port, int backlog)`
- `public ServerSocket (int port, int backlog, InetAddress bindAddr)`
 - `port`: puerto de la máquina servidora.
 - `backlog`: tamaño de la cola de espera, en el primero es 50.
 - `bindAddr`: dirección IP local que se hará pública mediante el bind.

- El constructor `ServerSocket` se encarga de hacer el bind y el listen.

Servicios de la clase ServerSocket

- `public Socket accept():`
 - Devuelve el socket resultado de aceptar una petición, para llevar a cabo la comunicación con el cliente.
- `public void close()`
 - Cierra el canal de comunicación.
- `public InetAddress getInetAddress():`
 - Devuelve la dirección IP de la máquina local.
- `public int getLocalPort():`
 - Devuelve el puerto de la máquina local.

Comunicación entre cliente-servidor

- Para la transmisión de datos entre cliente y servidor se utilizarán las clases `DataInputStream` (recibir datos) y `DataOutputStream` (enviar datos)
- Estas clases disponen de métodos para leer y escribir datos en el socket:
 - `read/writeBoolean`
 - `read/writeChar`
 - `read/writeDouble`, `read/writeFloat`, `read/writeInt`, `read/writeLong`, `read/writeShort`
 - `read/writeUTF` (leer/escribir cadenas de caracteres)
- Para enviar los datos se utiliza el método `flush()` de la clase `DataOutputStream`.

Clase DatagramSocket

- Constructores:

- `public DatagramSocket ()`
- `public DatagramSocket (int port)`
- `public DatagramSocket (int port, InetAddress laddr)`

- port: puerto de la máquina.
- laddr: dirección IP local que se hará pública mediante el bind.

- El constructor `DatagramSocket` se encarga de hacer el bind.
- El primer constructor coge un puerto libre.

Servicios de la clase DatagramSocket

- public void connect(InetAddress address, int port):
 - Conecta el socket a la máquina remota con la @IP address y puerto port.
- public void close()
 - Cierra el canal de comunicación.
- public void disconnect():
 - Desconecta el socket.
- public InetAddress getInetAddress():
 - Devuelve la @IP de la máquina remota.
- public int getPort():
 - Devuelve el puerto de la máquina remota.
- Tambien hay dos llamadas para saber la @IP y puerto local (getLocalAddress() y getLocalPort()).

Servicios de la clase DatagramSocket

- `public void send(DatagramPacket p):`
 - Envía un datagrama a la máquina remota, por el socket asociado.
- `public void receive(DatagramPacket p):`
 - Recibe un datagrama de otra máquina, por el socket asociado.

Clase DatagramPacket

• Constructores:

- `public DatagramPacket (byte[] buff, int length)`
 - Construye un `DatagramPacket` para recibir paquetes en el buffer `buff`, de longitud `length`
- `public DatagramPacket (byte[] buff, int length, InetAddress address, int port)`
 - Construye un `DatagramPacket` para enviar paquetes con datos del buffer `buff`, de longitud `length`, a la @IP `address` y el puerto `port`.

• Servicios:

- Para la actualización y consulta de los diferentes campos de un `DatagramPacket` disponemos de los siguientes métodos: `set/getAddress`, `set/getData`, `set/getLength`, `set/getPort`

A spiral-bound notebook with a brown cover and a white page. The spiral binding is on the left side. The word "Threads" is written in a large, black, serif font in the center of the page.

Threads

Qué es un thread?

- Definición: Un thread es un flujo secuencial de control dentro de un programa.
- Está definido por un contador de programa (PC) y un puntero a la pila (SP)
- Un thread no puede ejecutarse por sí mismo, ha de hacerlo dentro de un programa.
- Todos los threads de un proceso comparten los recursos (Ej: canales E/S, memoria, ...)

Qué ganamos?

- Conseguimos concurrencia entre procesos y también dentro de un mismo proceso.
- También podemos conseguir paralelismo si disponemos de una arquitectura multiprocesador.
- El cambio de contexto entre threads no es tan costoso al sistema como el cambio de contexto entre procesos.

Atributos de un thread en JAVA

- Un thread tiene:
 - Nombre
 - Prioridad
 - Ha de seguir la siguiente relación:
 - $\text{MIN_PRIORITY} \leq \text{Prioridad} \leq \text{MAX_PRIORITY}$
 - La prioridad por defecto es `NORM_PRIORITY`
 - Valores:
 - `MIN_PRIORITY = 1`
 - `NORM_PRIORITY = 5`
 - `MAX_PRIORITY = 10`

Clase Thread

- Constructores:

- `public Thread ()`
 - Crea un thread estándar.
- `public Thread (Runnable target)`
 - Crea un thread en base a una clase que implementa la interfaz `Runnable`.
- `public Thread (Runnable target, String name)`
 - Equivalente al anterior constructor dándole el nombre `name` al thread.
- `public Thread (String name)`
 - Crea un thread estándar con el nombre `name`.

Clase Thread

- `public Thread (ThreadGroup group, Runnable target)`
 - Crea un thread en base a una clase que implementa la interfaz `Runnable` dentro del `ThreadGroup`.
- `public Thread (ThreadGroup group, Runnable target, String name)`
 - Equivalente al anterior constructor dándole el nombre `name` al thread.
- `public Thread (ThreadGroup group, String name)`
 - Crea un thread estándar con el nombre `name` dentro del `ThreadGroup`
- Cuando se crea un thread no se pone en marcha, se lo hemos de indicar explícitamente.
- Un `ThreadGroup` es un conjunto de threads, los cuales tienen acceso a información entre ellos.

Servicios de la clase Thread

- `public String getName ():`
 - Devuelve el nombre que identifica al thread.
- `public String setName (String name):`
 - Cambia el nombre del thread.
- `public int getPriority()`
 - Devuelve la prioridad del thread.
- `public void setPriority(int newPriority)`
 - Cambia la prioridad del thread
- `public ThreadGroup getThreadGroup():`
 - Devuelve el grupo de donde procede el thread.

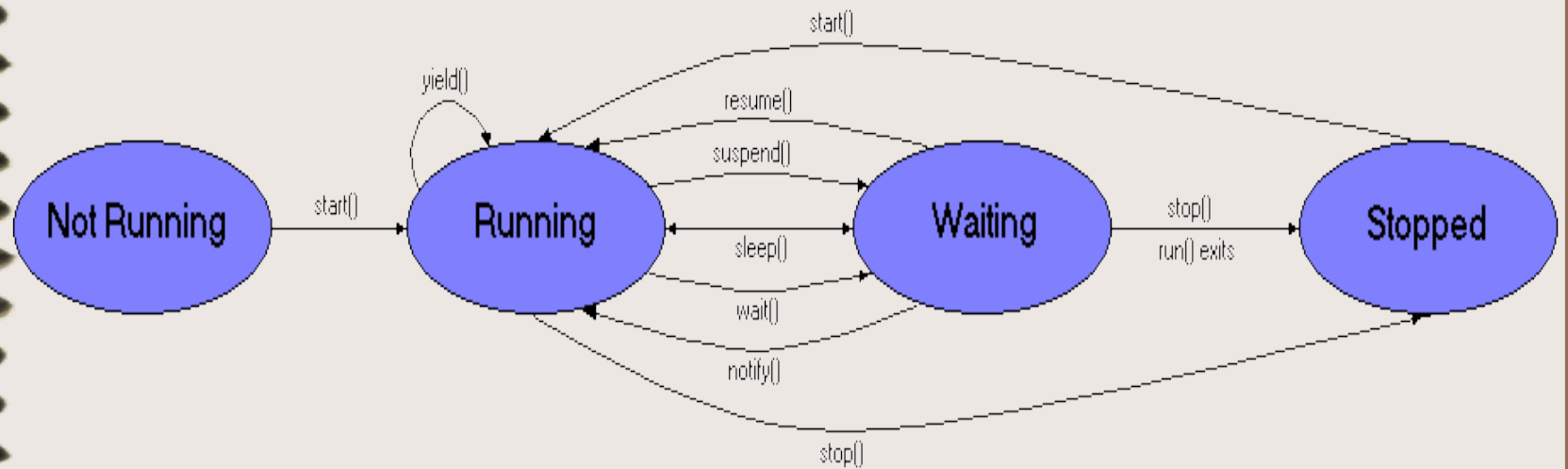
Servicios de la clase Thread

- void run()
 - Es el método que ejecutará un thread cuando este se ponga en marcha.
- void start()
 - Inicia la ejecución del thread.
- void join()
 - Espera a que un thread termine su ejecución.
 - Podemos especificar el tiempo de espera máximo.
- void sleep(long milis)
 - Bloquea el thread milis milisegundos.
 - Hay otro parámetro que permite afinar más el tiempo de bloqueo en nanosegundos.

Servicios de la clase Thread

- void stop/destroy()
 - Destruye el thread sin eliminar estructuras, esto es tarea del método join.
- static Thread currentThread()
 - Devuelve el thread que se está ejecutando actualmente.
- boolean isAlive()
 - Mira si un thread no ha terminado su ejecución
- static void yield()
 - Provoca el cambio de contexto entre flujos.

Estados de un thread



Sincronización

- Aquí se presenta el problema de que varios threads intenten acceder a los mismos datos y que el resultado final sea incorrecto.
 - Por ejemplo: Acceso a una variable global x con valor inicial 0, y dos threads que intentan incrementar su valor en una unidad.
 - Si los threads no están sincronizados un resultado posible sería que la variable x tenga el valor 1 (incorrecto)

Sincronización

- Solución: Para resolver el problema anterior se utiliza la etiqueta `synchronized` junto con el objeto que se quiere sincronizar.
 - En el caso anterior se solucionaría de la siguiente forma:
 - `synchronized (this) { x++; }`
 - `this` es la clase que contiene la variable global
 - Con esto se garantiza el acceso en exclusión mutua.

Sincronización

- Si queremos acceder a un método en exclusión mutua tenemos las siguientes opciones:
 - Igual que el ejemplo anterior:
 - `synchronized (this) { this.incrementar() }`
 - Declarando el método como sincronizado:
 - `synchronized void incrementar ()`
- También se pueden sincronizar varios threads mediante los métodos: `wait()`, `notify()` y `notifyAll()`

A spiral-bound notebook with a brown cover and a white page. The spiral binding is on the left side. The text "Ejemplo: Un chat" is written in the center of the page in a black serif font.

Ejemplo:
Un chat

Servidor.java (I)

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

public class Servidor extends Thread
{
    public static Vector usuarios = new Vector();
    public static void main (String args[])
    {
        ServerSocket sfd = null;
        try
        {
            sfd = new ServerSocket(7000);
        }
        catch (IOException ioe)
        {
            System.out.println("Comunicación rechazada."+ioe);
            System.exit(1);
        }
    }
}
```

Servidor.java (II)

```
while (true)
{
    try
    {
        Socket nsfd = sfd.accept();
        System.out.println("Conexion aceptada de: "+nsfd.getInetAddress());
        Flujo flujo = new Flujo(nsfd);
        Thread t = new Thread(flujo);
        t.start();
    }
    catch(IOException ioe)
    {
        System.out.println("Error: "+ioe);
    }
}
}
```

Flujo.java (I)

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

```
public class Flujo extends Thread  
{  
    Socket nsfd;  
    DataInputStream FlujoLectura;  
    DataOutputStream FlujoEscritura;
```

```
public Flujo (Socket sfd)  
{  
    nsfd = sfd;  
    try  
    {  
        FlujoLectura = new DataInputStream(new BufferedInputStream(sfd.getInputStream()));  
        FlujoEscritura = new DataOutputStream(new BufferedOutputStream(sfd.getOutputStream()));  
    }  
}
```

Flujo.java (II)

```
catch(IOException ioe)
{
    System.out.println("IOException(Flujo): "+ioe);
}
}

public void run()
{
    broadcast(nsfd.getInetAddress()+"> se ha conectado");
    Servidor.usuarios.add ((Object) this);
    while(true)
    {
        try
        {
            String linea = FlujoLectura.readUTF();
            if (!linea.equals(""))
            {
                linea = nsfd.getInetAddress() + ">> "+ linea;
                broadcast(linea);
            }
        }
    }
}
```


Flujo.java (III)

```
catch(IOException ioe)
{
    Servidor.usuarios.removeElement(this);
    broadcast(nsfd.getInetAddress()+"> se ha desconectado");
    break;
}
}
```

```
public void broadcast(String mensaje)
{
    synchronized (Servidor.usuarios)
    {
        Enumeration e = Servidor.usuarios.elements();
        while (e.hasMoreElements())
        {
            Flujo f = (Flujo) e.nextElement();
```

Flujo.java (IV)

```
try
{
    synchronized(f.FlujoEscritura)
    {
        f.FlujoEscritura.writeUTF(mensaje);
        f.FlujoEscritura.flush();
    }
}
catch(IOException ioe)
{
    System.out.println("Error: "+ioe);
}
}
}
```

Cliente.java (I)

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

public class Cliente extends Frame implements ActionListener
{
    static Socket sfd = null;
    static DataInputStream EntradaSocket;
    static DataOutputStream SalidaSocket;
    static TextField salida;
    static TextArea entrada;
    String texto;

    public Cliente()
    {
        setTitle("Chat");
        setSize(350,200);
    }
}
```

Cliente.java (II)

```
salida = new TextField(30);  
salida.addActionListener(this);
```

```
entrada = new TextArea();  
entrada.setEditable(false);
```

```
add("South",salida);  
add("Center", entrada);  
setVisible(true);
```

```
}
```

```
public static void main(String[] args)
```

```
{
```

```
    Cliente cliente = new Cliente();
```

```
    try
```

```
    {
```

```
        sfd = new Socket("192.168.0.1",7000);
```

```
        EntradaSocket = new DataInputStream(new BufferedInputStream(sfd.getInputStream()));
```

```
        SalidaSocket = new DataOutputStream(new BufferedOutputStream(sfd.getOutputStream()));
```

```
    }
```

Cliente.java (III)

```
catch (UnknownHostException uhe)
{
    System.out.println("No se puede acceder al servidor.");
    System.exit(1);
}
catch (IOException ioe)
{
    System.out.println("Comunicación rechazada.");
    System.exit(1);
}
while (true)
{
    try
    {
        String linea = EntradaSocket.readUTF();
        entrada.append(linea+"\n");
    }
    catch(IOException ioe)
    {
        System.exit(1);
    }
}
```

Cliente.java (IV)

```
}  
}  
  
public void actionPerformed (ActionEvent e)  
{  
    texto = salida.getText();  
    salida.setText("");  
    try  
    {  
        SalidaSocket.writeUTF(texto);  
        SalidaSocket.flush();  
    }  
    catch (IOException ioe)  
    {  
        System.out.println("Error: "+ioe);  
    }  
}  
  
public boolean handleEvent(Event e)  
{
```

Cliente.java (V)

```
if ((e.target == this) && (e.id == Event.WINDOW_DESTROY))
{
    if (sfd != null)
    {
        try
        {
            sfd.close();
        }
        catch (IOException ioe)
        {
            System.out.println("Error: "+ioe);
        }
        this.dispose();
    }
}
return true;
}
```

Bibliografía

- Dossier de la asignatura CASO
- “JAVA El lenguaje de programación de Internet”
Ed.Data Becker 1995
- Tutoriales de JAVA extraídos de:
 - <http://fibers.upc.es/pages/tutorials.php>
- API de JAVA versión 1.3