Tema 1 (Parte 2)
Servidor TCP Java

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15

Índice

Clases Java

Socket

Representa un canal de comunicación con otra máquina.

Hay que obtener un objeto de este tipo en cada máquina.

Una vez se ha obtenido se usan las clases de entrada/salida de Java (`InputStream`, `OutputStream`, `DataInputStream`, `DataOutputStream`, `PrintWriter`, `BufferedReader`, `...` para pasar información de una máquina a otra.

La conexión se mantiene abierta hasta que se cierre explícitamente o se alcance un tiempo (configurable) sin obtener información.

Clases Java

ServerSocket

Un objeto de este tipo se pone a la escucha en un puerto de la máquina donde se crea.

Permite recibir peticiones de conexión de los clientes.

Su método `accept()` devuelve una referencia del tipo `Socket` que permite la comunicación con el cliente.

Para no aceptar más conexiones se usa el método `close()`.

Ejemplo de servidor

```
class ServidorTCP{
   public static void main(String[] args) {
       ServerSocket s = new ServerSocket(8080);
       while (true){
          try{
             Socket canal = s.accept();
             InputStream in = canal.getInputStream();
             OutputStream out = canal.getOutputStream();
             // Lectura/Escritura de la información
             in.close();
             out.close();
             canal.close();
          }catch(Exception ex){
             ex.printStackTrace();
          }
       }
   }
}
```


Ejemplo de cliente

```
class ClienteTCP{
   public static void main(String[] args) {
      String serverHost = ...;
      try{
         Socket canal = new Socket(serverHost,8080);
         InputStream in = canal.getInputStream();
         OutputStream out = canal.getOutputStream();
         // Lectura/Escritura de la información
         in.close();
         out.close();
         canal.close();
      }catch(Exception ex){
         ex.printStackTrace();
      }
   }
}
```


Problemas

Servidor que acepta peticiones concurrentes

Clase `Thread`

```
class Thread{
   Thread(){...}
   Thread(Runnable r){...}
   public void run(){}
   public void start(){...}
   public void join(){...}
   public void sleep(long milis) throws InterruptedException {...}
   public void yield(){...}
   ...
}
```

Servidor que acepta peticiones concurrentes

Interface `Runnable`

```
public interface Runnable{
   public void run()
}
```

Patrón para ejecutar código concurrentemente definido en una clase anónima que implementa a `Runnable`

```
...
new Thread(new Runnable(){
   public void run(){
      // Código
   }
}).start();
...
```


Servidor que acepta peticiones concurrentes

```
class TrataConexion implements Runnable{
   private Socket canal;
   TrataConexion(Socket s){
      canal = s;
   }
   public void run(){
      try{
         InputStream in = canal.getInputStream();
         OutputStream out = canal.getOutputStream();
         // Lectura/Escritura de la información
         in.close();
         out.close();
         canal.close();
      }catch(Exception ex){
         ex.printStackTrace();
      }
   }
}
```


Servidor que acepta peticiones concurrentes

```
class ServidorTCP{
   public static void main(String[] args) {
       ServerSocket s = new ServerSocket(8080);
       while (true){
          try{
             Socket canal = s.accept();
             new Thread(new TrataConexion(canal)).start();
          }catch(Exception ex){
             ex.printStackTrace();
          }
       }
   }
}
```


Problemas

¿Qué problemas presenta el código del servidor anterior?

Servidor que acepta un número máximo de peticiones concurrentes

La clase `java.util.concurrent.Executors` ofrece una serie de métodos estáticos que permiten crear un pool de hilos.

En concreto:

```
ExecutorService ex = Executors.newFixedThreadPool(nThreads);
```

Una vez se obtiene un objeto de este tipo se pueden ejecutar tareas usando el método:

```
class ExecutorService{
   public void execute(Runnable r){...}
}
```


Servidor que acepta un número máximo de peticiones concurrentes

```
ServerSocket s = new ServerSocket(port);

ExecutorService ex = Executors.newFixedThreadPool(nThreads);

while (true) {
   try{
      Socket canal = s.accept();
      Runnable handleRequest = new HiloCliente(canal, path, priv,
            accessFile);
      ex.execute(handleRequest);
   }catch(Exception ex){
   }
}
```