Aplicaciones Client/Server Usando Sockets de Java

Client/Server Applications Using Java Sockets

Samir Genaim

Ejemplo de un Servidor Sencillo

```
void startServer() throws IOException {
    ServerSocket server = new ServerSocket(2000);

    while ( ... ) {
        Socket s = server.accept();
        handleRequest(s);
    }

    server.close();
}

void handleRequest(Socket s) throws IOException {
    ...
    Scanner in = new Scanner( s.getInputStream() );

    do {
        i = in.nextInt();
        System.out.println("Recevied: "+i);
    } while (i != -1);
}
```

Crear un servidor que escucha la puerto 2000

La llamada a accept() bloquea hasta que alguien se conecta, en ese caso devuelve un socket que se puede utilizar para enviar/recibir datos

manejar la petición

Apagar el servidor

Obtener un InputStream que para leer los datos enviados por el cliente. Es un stream de bytes, así que mejor pasarlo a un Scanner para usarlo fácilmente - esto depende de los datos que está esperando.

Leer un entero enviado por el cliente

see: examples.clientserver.ex1.Server

Ejemplo de un Cliente Sencillo

```
void sendToServer() throws ... {
    Random r = new Random();

    Socket s = new Socket("localhost", 2000);

    PrintStream p = new PrintStream( s.getOutputStream() );

    for(int i=0; i<10; i++) {
        p.println( r.nextInt(1000) );
        p.flush();
        sleep(300);
    }

    p.print(-1);
    s.close();
}
```

Conectar a " localhost " en el puerto 2000. El Socket se puede utilizar para enviar/recibir datos a/desde el servidor.

Obtener un OutputStream que para enviar datos al servidor. Es un stream de bytes, así que mejor pasarlo a un PrintStream para usarlo fácilmente - esto depende de los datos que vamos a enviar.

Enviar un entero

"Flush" el stream para enviar los datos inmediatamente

Cerrar el socket

see: examples.clientserver.ex1.Client

Enviar Datos al Cliente

```
void handleRequest(Socket s) throws IOException {
    ...
    PrintStream out = new PrintStream(s.getOutputStream());
    Scanner in = new Scanner(s.getInputStream());

    do {
        i = in.nextInt();
        System.out.println("Recevied: "+i);
        out.println(2+"*"+i+"="+(2*i));
    } while (i != -1);

}
```

Crear canales de entrada y salida con el cliente

Leer un número enviado por el cliente ...

... y envíale algo

```
void sendToServer() throws ... {
    ...
    Socket s = new Socket("localhost", 2000);
    PrintStream p = new PrintStream(s.getOutputStream());
    Scanner in = new Scanner(s.getInputStream());

    for (int i = 0; i < 10; i++) {
        p.println(r.nextInt(1000));
        p.flush();
        System.out.println(in.nextLine());
        sleep(300);
    }
}
```

Crear canales de entrada y salida con el servidor

Enviar un entero al servidos ...

... y recibir una respuesta.

see: examples.clientserver.ex2

Atender a Clientes en Paralelo

```
void startServer() throws IOException {
    ...
    while ( ... ) {
        Socket s = server.accept();
        handleRequestInThread(s);
    }
    ...
}

void handleRequestInThread(Socket s) {
    new Thread() {
        public void run() {
            try {
                handleRequest(s);
            } catch (IOException e) {
            }
        }
        }.start();
}
```

Para atender a los clientes de forma paralela creamos una hebra para manejar la petición ...

..., es decir, crear una hebra que llama a handleRequest

see: examples.clientserver.ex3.Server

Atender a Clientes en Paralelo

Usando Executer

```
// Declare the executer as a static field in the class
// Executor exec = Executors.newSingleThreadExecutor();
// Executor exec = Executors.newFixedThreadPool(10);
Executor exec = Executors.newCachedThreadPool();

void handleRequestInThread(Socket s) {
    exec.execute(new Runnable() {

        @Override
        public void run() {
            try {
                handleRequest(s);
            } catch (IOException e) {
            }
        }
    });
}
```

Un executer es un "woker" que mantiene una o más hebras. Recibe tareas (como Runnable) y las ejecuta en esas hebras. Las tareas van primero a una cola y serán ejecutadas cuando cuando hay hebras "libres" ...

CUIDADO: si el Executer tiene un número finito/fijo de hebras, puede ser que el servidor no se atiende a unos clientes si la conexión mantiene durante mucho tiempo ...

see: examples.clientserver.ex3.ServerWithExecuter

Controlar el Servidor

- ¿Cómo podemos controlar el servidor mientras está ejecutando? por ejemplo pararlo, consultar información sobre su estado, etc.
- Podemos ejecutar el servidor en una hebra y otro código de control en otra ...

```
public void launchServer() throws ... {
    startSertverInAThread();
    control();
}
```

```
private void startSertverInAThread() {
    new Thread() {
        @Override
        public void run() {
            try { startServer(); } catch (IOException e) { ... }
        }
    }.start();
}
```

see: examples.clientserver.ex4

Controlar el Servidor

```
void control() throws IOException {
    Scanner in = new Scanner(System.in);
    while (!stopped) {
        System.out.print("...");
        String cmd = in.nextLine();
        switch (cmd) {
        case "status":
            ...
            break;
        case "stop":
            stopped = true;
            server.close();
        default:
            break;
        }
    }
}
```

```
void startServer() throws ... {
    ...
    stopped = false;
    while (!stopped) {
        try {
            s = server.accept();
            ...
        } catch (IOException e) {
        }
    }
    ...
}
```

- `stopped` es un atributo compartido, decláralo `volatile`
- startServer y control están ejecutando en dos hebras
- cuando 'control' ejecuta `server.close()`, la instrucción `server.accept()` del 'startServer' lanza una excepción si está bloqueada esperando una conexión
- el método control puede crear un ventana en lugar de usar la consola ...

see: examples.clientserver.ex4

¿Cómo Enviar y Recibir Objetos?

- Sólo usa ObjectInputStream y ObjectOutputStream en lugar de Scanner y PrintStream ...
- Vamos a cambiar el servidor y cliente para que envíen/reciban objetos de tipo MyNumber ...

```
public class MyNumber implements Serializable {

    private static final long serialVersionUID = 1L;
    private int n;

    public MyNumber(int n) {
        this.n = n;
    }

    public int getValue() {
        return n;
    }
}
```

see: examples.clientserver.ex5

Enviar/Recibir MyNumber: Servidor

```
private void handleRequest(Socket s) throws ... {
    ...
    ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
    ObjectInputStream in = new ObjectInputStream(s.getInputStream());

    do {
        i = ((MyNumber)in.readObject()).getValue();
        if ( i != -1) {
            out.writeObject( new MyNumber(2*i) );
            out.flush();
            out.reset();
        }
    } while (i != -1);
      ...
}
```

Creamos los canales de comunicación usando los streams de objetos.

Recibir un objeto y hacer casting a MyNumber

Crear una instancia de MyNumber y enviarla al cliente

Usar a flush() y reset() después de enviar el objeto para asegurarse de que se transmite inmediatamente.

see: examples.clientserver.ex5

Enviar/Recibir MyNumber: Cliente

Creamos los canales de comunicación usando los streams de objetos.

```
public static void sendToServer() throws ... {
    ...
    Socket s = new Socket("localhost", 2000);
    ObjectOutputStream p = new ObjectOutputStream(s.getOutputStream());
    ObjectInputStream in = new ObjectInputStream(s.getInputStream());

    for (int i = 0; i < 10; i++) {
        p.writeObject( new MyNumber(r.nextInt(1000)));
        p.flush();
        p.reset();
        MyNumber n = (MyNumber) in.readObject();
         ...
        sleep(300);
    }
     ...
}
```

Enviar un objeto de tipo MyNumber.

Usar a flush() y reset() después de enviar el objeto para asegurarse de que se transmite inmediatamente.

Recibir un objeto y hacer casting a MyNumber

see: examples.clientserver.ex5

Importante

```
Socket s
…
ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
ObjectInputStream in = new ObjectInputStream(s.getInputStream());
```

- El orden en el que se ejecuta s.getOutputStream() y s.getInputStream( ) puede ser importante …
- Si primero llamas as.getInputStream(), la ejecución podría bloquear hasta que el otro lado del socket envia algunos datos …
- Siempre llamar a s.getOutputStream() primero si ambos son necesarios