# **Construcción de un servidor web *multi-hilos***

## **Servidor Web en lenguaje Java: Parte I**

### **Servidor web *multi-hilos* que mostrará el contenido de la solicitud (request) HTTP que recibe.**

A continuación se hará un recorrido a través del código para la primera parte de la implementación de un servidor Web. En los lugares donde usted encuentre un signo de interrogación, **?**, usted debe completar el código que hace falta.

La primera parte de la implementación del servidor web será *multi-hilos*, ya que el procesamiento de cada solicitud HTTP que llegue será atendida dentro de un *hilo de ejecución* separado. Esto permitirá al servidor:

* Atender múltiples clientes en paralelo o
* Realizar transferencias de archivos múltiples en paralelo al mismo cliente.

Cuando se crea un nuevo *hilo* de ejecución, se debe pasar al constructor de hilos una instancia de alguna clase que implemente la interface Runnable. Esta es la razón por la cual se debe definir una clase separada llamada SolicitudHttp. La estructura del servidor web será la siguiente:

import java.io.\* ;

import java.net.\* ;

import java.util.\* ;

public final class ServidorWeb

{

public static void main(String argv[]) throws Exception

{

. . .

}

}

final class SolicitudHttp implements Runnable

{

. . .

}

Normalmente, los servidores web atienden las solicitudes que ellos reciben a través del puerto 80. Para este práctica usted puede escoger cualquier número de puerto superior a 1024 (pero recuerde utilizar este mismo número cuando haga la solicitud desde el *browser*). En el siguiente trozo de código se utiliza el número 6789 para el puerto.

public static void main(String argv[]) throws Exception

{

// Establece el número de puerto.

int puerto = 6789;

. . .

}

Luego, se abre un socket y se espera a que se solicite una conexión TCP. Debido a que el servidor estará recibiendo solicitudes indefinidamente, se colocará la operación de escucha (*listen*) dentro de un ciclo infinito. Esto significa que el servidor web debe ser terminado presionando al mismo tiempo en el teclado las teclas Control y C (^C).

// Estableciendo el socket de escucha.

**?**

// Procesando las solicitudes HTTP en un ciclo infinito.

while (true) {

// Escuhando las solicitudes de conexión TCP.

**?**

. . .

}

Cuando se reciba una solicitud de conexión se debe crear un objeto SolicitudHttp, pasando al método constructor una referencia al objeto Socket que representa la conexión establecida con el cliente.

// Construye un objeto para procesar el mensaje de solicitud HTTP.

SolicitudHttp solicitud = new SolicitudHttp( **?** );

// Crea un nuevo hilo para procesar la solicitud.

Thread hilo = new Thread(solicitud);

// Inicia el hilo.

hilo.start();

Para que el objeto SolicitudHttp maneje la solicitud de servicio HTTP que llega en un hilo separado, primero se debe crear una nueva instancia del objetoThread, pasando a su método constructor una referencia al objeto SolicitudHttp, y luego se invoca el método start() del hilo recien creado.

Después que el nuevo hilo ha sido creado e iniciado, la ejecución en el hilo principal regresa al inicio del ciclo de procesamiento del mensaje. Entonces el hilo principal podrá seguir *escuchando*, esperando otras solicitudes de conexión TCP, mientras el nuevo hilo continua corriendo. Cuando se reciba otra solicitud de conexión TCP, el hilo principal realizará el mismo proceso de creación de un nuevo hilo sin importar si el hilo previo ha terminado o aún continua su ejecución.

Esto completa el código del método main(). Para el resto de la primera parte de la práctica, sólo resta desarrollar el código de la clase SolicitudHttp.

Se declararán dos variables para la clase SolicitudHttp: CRLF y socket. De acuerdo con la especificación del protocolo HTTP, debemos terminar cada línea de los mensajes de respuesta del servidor con un *carriage return* (CR) y un *line feed* (LF), por esto es conveniente definir CRLF. La variable socket será utilizada para guardar una referencia al socket de conexión, con el cual se invocó el constructor de esta clase. La estructura de la clase SolicitudHttp es mostrada a continuación:

final class SolicitudHttp implements Runnable

{

final static String CRLF = "\r\n";

Socket socket;

// Constructor

public SolicitudHttp(Socket socket) throws Exception

{

this.socket = socket;

}

// Implementa el método run() de la interface Runnable.

public void run()

{

. . .

}

private void proceseSolicitud() throws Exception

{

. . .

}

}

Para poder pasar una instancia de la clase SolicitudHttp al constructor de hilos, SolicitudHttp debe implementar la interface Runnable, que simplemente significa que se debe definir un método público llamado run() que retorna void. La mayor parte del procesamiento se realizará dentro del método proceseSolicitud(), que es invocado desde run().

Hasta este punto se ha venido permitiendo lanzar excepciones, sin manejarlas. Sin embargo, no se pueden lanzar excepciones desde run(), ya que se debe respetar estrictamente la declaración de run() en la interface Runnable, que no lanza ninguna excepción. Todo el código para procesamiento será colocado en proceseSolicitud(), y desde allí, lanzar las excepciones para run(). Dentro de run(), de forma explícita se capturarán y se manejarán las excepciones con un bloque try/catch.

// Implementa el método run() de la interface Runnable.

public void run()

{

try {

proceseSolicitud();

} catch (Exception e) {

System.out.println(e);

}

}

El siguiente paso es desarrollar el código interno de proceseSolicitud(). Primero se obtiene una referencia al stream de salida del socket. Luego se obtiene una referencia al stream de entrada del socket y se envuelven los filtros InputStreamReader y BufferedReader alrededor del stream de entrada ( no se debe envolver ningún filtro alrededor del stream de salida, ya que se escribirá directamente en el stream de salida).

private void proceseSolicitud() throws Exception

{

// Referencia al stream de salida del socket.

DataOutputStream os = **?**;

// Referencia y filtros (InputStreamReader y BufferedReader)para el stream de entrada.

BufferedReader br = **?**;

. . .

}

Ahora se puede recoger el mensaje de solicitud HTTP del cliente, esto se hace leyendo el stream de entrada del socket. El método readLine() de la clase BufferedReader extrae caracteres del stream de entrada hasta que encuentre un caracter end-of-line, o en este caso, la secuencia CRLF.

El primer elemento disponible en el stream de entrada será la línea del mensaje de solicitud HTTP.

// Recoge la línea de solicitud HTTP del mensaje.

String lineaDeSolicitud = **?**;

// Muestra la línea de solicitud en la pantalla.

System.out.println();

System.out.println(lineaDeSolicitud);

Después de obtener la línea de solicitud del mensaje, se deben recoger las líneas del header. Ya que no se sabe con anticipación cuántas líneas de header enviará el cliente, estas se deben recoger mediante un ciclo.

// recoge y muestra las líneas de header.

String lineaDelHeader = null;

while ((lineaDelHeader = br.readLine()).length() != 0) {

System.out.println(lineaDelHeader);

}

Por ahora no se requiere más de las líneas del header: sólo que sean mostradas en la pantalla, esto se puede hacer utilizando una variable tipo String, lineaDelHeader, para tener una referencia a los valores de estas líneas. El ciclo termina cuando la expresión

(lineaDelHeader = br.readLine()).length()

evalua un cero, que ocurrirá cuando la variable lineaDelHeader tenga una longitud cero. Esto sucederá cuando la línea quede vacia, terminando la lectura de las líneas de header. Faltaría colocar las siguientes líneas de código para cerrar los streams y el socket de conexión.

// Cierra los streams y el socket.

os.close();

br.close();

socket.close();

En la [siguiente parte de la práctica](http://www.arcesio.net/practicas/WebServer2.htm) (Parte II), se adicionará el código para analizar el mensaje de solicitud del cliente y se enviará una respuesta HTTP bien construida. Pero antes de hacer eso, compile el código que tiene hasta el momento. Después que su programa compile de manera correcta, ejecútelo con un número de puerto disponible y trate de contactarlo desde un browser para web. Para hacer esto, usted debe utilizar la dirección IP o el nombre del equipo donde está ejecutando el servidor y el número de puerto utilizado. Por ejemplo, si su máquina se llama www.miservidor.com y usted colocó el servidor en el puerto 6789, entonces debe especificar el siguiente URL:

http://www.miservidor.com:6789/

El servidor debe mostrar el contenido del mensaje de solicitud HTTP.

* [Construcción de un servidor web *multi-hilos*](http://www.arcesio.net/practicas/WebServer.htm)
* Servidor Web en lenguaje Java: Parte I (servidor web *multi-hilos* que mostrará el contenido de la solicitud (request) HTTP que recibe).
* [Servidor Web en lenguaje Java: Parte II (servidor web *multi-hilos* que genera una respuesta HTTP apropiada)](http://www.arcesio.net/practicas/WebServer2.htm).
* [Servidor web simple de ejemplo (no es *multi-hilos*)](http://www.arcesio.net/practicas/ServidorWebSimple.txt)