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

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

### **Servidor web *multi-hilos* que genera una respuesta HTTP apropiada**

En esta seguna parte, en lugar de terminar el hilo después de mostrar en pantalla el mensaje de solicitud HTTP enviado por el browser, se analizará el mensaje de solicitud HTTP y se enviará una respuesta HTTP apropiada al browser. Se ignorará la información de las líneas de header y se se utilizará sólo el nombre del archivo que viene en la línea de solicitud. De hecho, se asume que el mensaje de solicitud HTTP siempre utilizará el método GET, e ignorará que el cliente puede enviar otros métodos HTTP, como HEAD ó POST. En los lugares donde usted encuentre un signo de interrogación, **?**, usted debe completar el código que hace falta.

El nombre del archivo de la línea de solicitud HTTP se extraerá con la ayuda de la clase java StringTokenizer. Primero se creará un objeto StringTokenizer que contendrá la cadena de caracteres de la línea de solicitud. Después no se tendrá en cuenta qué metodo se especificó en la solicitud HTTP, pues se asumirá que sólo puede ser el método GET. Finalmente se extraerá el nombre del archivo.

// Extrae el nombre del archivo de la línea de solicitud.

StringTokenizer partesLinea = new StringTokenizer(lineaDeSolicitud);

partesLinea.nextToken(); // "salta" sobre el método, se supone que debe ser "GET"

String nombreArchivo = partesLinea.nextToken();

// Anexa un ".", de tal forma que el archivo solicitado debe estar en el directorio actual.

nombreArchivo = "." + nombreArchivo;

Ya que el browser precede el nombre del archivo con un slash, "/", se colocará antes un punto de tal forma que el nombre de path indicará que el archivo se encuentra dentro del directorio actual.

Ahora que se tiene el nombre del archivo, éste se puede abrir para enviarlo al cliente. Si el archivo solicitado no existiera, el constructor FileInputStream() lanzaría la excepción FileNotFoundException. Pero en lugar de lanzar la excepción y terminar el hilo, se utilizará una pareja try/catch para colocar la variable booleana existeArchivo en *false* (falso). Posteriormente, en el código el valor de esta variable se utilizará para construir un mensaje de respuesta indicando un error, en lugar de tratar de enviar un archivo que no existe.

// Abre el archivo seleccionado.

FileInputStream fis = null;

boolean existeArchivo = true;

try {

fis = new FileInputStream(nombreArchivo);

} catch (FileNotFoundException e) {

existeArchivo = false;

}

El mensaje de respuesta HTTP tiene tres partes: la línea de estado, los headers de respuesta y el cuerpo o datos que lleva el mensaje. La línea de estado y los headers de respuesta terminan con la secuencia de caracteres CRLF. Se responderá con una línea de estado, que se colocará en la variable lineaDeEstado, y un único header de respuesta, que se guardará en la variable lineaDeTipoContenido. En el caso de que se solicite un archivo que no existe, se retornará la línea "*404 Not Found*" en la línea de estado del mensaje de respuesta HTTP y se incluirá, en el cuerpo del mensaje, un error descrito en un documento HTML.

// Construye el mensaje de respuesta.

String lineaDeEstado = null;

String lineaDeTipoContenido = null;

String cuerpoMensaje = null;

if (existeArchivo) {

lineaDeEstado = **?**;

lineaDeTipoContenido = "Content-type: " +

contentType( nombreArchivo ) + CRLF;

} else {

lineaDeEstado = **?**;

lineaDeTipoContenido = **?**;

cuerpoMensaje = "<HTML>" +

"<HEAD><TITLE>404 Not Found</TITLE></HEAD>" +

"<BODY><b>404</b> Not Found</BODY></HTML>";

}

Cuando el archivo existe, se debe determinar el tipo de archivo MIME y enviar el especificador de tipo MIME apropiado. Esta determinación de tipo se realiza a través de un método privado y separado llamado contentType(), que retorna una cadena que puede incluirse en la línea de tipo de contenido (content type) que se está construyendo.

Ahora se puede enviar la línea de estado al browser escribiendo en el stream de salida asociado al socket.

// Envia la línea de estado.

os.writeBytes(lineaDeEstado);

// Envía el contenido de la línea content-type.

os.writeBytes(**?**);

// Envía una línea en blanco para indicar el final de las líneas de header.

os.writeBytes(CRLF);

Ya que la línea de estado y la línea de header terminadas con CRLF han sido colocadas en el stream de salida para ser enviadas al browser, es el momento de hacer lo mismo con el cuerpo del mensaje. Si el archivo solicitado existe, se invoca un método separado para enviar el archivo. Si el archivo solicitado no existe, se envía un menaje de error escrito en HTML que ya ha sido preparado antes.

// Envía el cuerpo del mensaje.

if (existeArchivo) {

enviarBytes(fis, os);

fis.close();

} else {

os.writeBytes(**?**);

}

Después de enviar el cuerpo, el trabajo de este hilo ha concluido, así que se deben cerrar los streams y el socket antes de terminar.

Aún se necesita codificar los dos métodos que se referenciaron en el anterior código, es decir, el método que determina el tipo MIME, contentType(), y el método que escribe el archivo solicitado en el stream de salida del socket. Primero se revisará el código para enviar el archivo al cliente.

private static void enviarBytes(FileInputStream fis, OutputStream os) throws Exception

{

// Construye un buffer de 1KB para guardar los bytes cuando van hacia el socket.

byte[] buffer = new byte[1024];

int bytes = 0;

// Copia el archivo solicitado hacia el output stream del socket.

while((bytes = fis.read(buffer)) != -1 ) {

os.write(buffer, 0, bytes);

}

}

Tanto read() como write() lanzan excepciones. En lugar de atraparlas y manejarlas en el código, serán lanzadas para que sean manejadas por el método que las invocó.

La variable, buffer, es un espacio de almacenamiento intermedio para bytes entre el archivo y el stream de salida. Cuando se leen los bytes desde FileInputStream, se revisa si read() retorna un menos uno, indicando que el final del archivo ha sido alcanzado. Si el final del archivo no ha sido alcanzado, read() retorna el número de bytes que han sido colocados en buffer. Se utiliza el método write()de la clase OutputStream para colocar estos bytes en el stream de salida, pasándole el nombre del arreglo de bytes, buffer, el punto de inicio del arreglo, 0, y el número de bytes en el arreglo que deben ser escritos, bytes.

La pieza final de código que se necesita para completar el servidor Web es un método que examinará la extensión del nombre del archivo y retornará una cadena que representa su tipo MIME. Si la extensión del archivo es desconocida, se retornará el tipo application/octet-stream.

private static String contentType(String nombreArchivo)

{

if(nombreArchivo.endsWith(".htm") || nombreArchivo.endsWith(".html")) {

return "text/html";

}

if(?) {

**?**;

}

if(?) {

**?**;

}

return "application/octet-stream";

}

En el trozo de código anterior faltan incluir nuevos tipos de archivos, como GIF y JPEG. Usted puede agregar los tipos de archivos que desee, de tal forma que los componentes de su página web sean enviados con el tipo de contenido correctamente especificado en la línea content type del header. Para el formato GIF el tipo MIME es image/gif y para el formatoJPEG es image/jpeg.

Esto completaría el código de la segunda parte de la práctica para desarrollar el servidor Web *multi-hilos*. Intente ejecutar el servidor desde el directorio donde su página está localizada y trate de ver la página con un browser. Recuerde específicar el puerto dentro del URL, para que el browser no intente conectarse al puerto 80. Cuando se conecte, examine en la pantalla las solicitudes que el servidor recibe del browser.

* [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)](http://www.arcesio.net/practicas/WebServer1.htm).
* Servidor Web en lenguaje Java: Parte II (servidor web *multi-hilos* que genera una respuesta HTTP apropiada).
* [Servidor web simple de ejemplo (no es *multi-hilos*)](http://www.arcesio.net/practicas/ServidorWebSimple.txt)