

Tema 1 (Parte 3)

Servidor estático HTTP

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15



En su versión básica un servidor estático HTTP debe:

- 1 leer el mensaje HTTP de petición,
- 2 obtener qué recurso se solicita,
- 3 obtener el recurso y
- 4 escribir el mensaje HTTP de respuesta con el recurso o un mensaje de error si el recurso no se encuentra.



- 1 [Servidor estático HTTP](#)
- 2 [Ejemplos de respuestas predefinidas](#)
- 3 [Configuración del servidor mediante un fichero de propiedades](#)



Normalmente añaden funcionalidad adicional:

- Proteger recursos con usuario y contraseña
- Posibilidad de configurar parámetros del servidor
- Almacenar información en ficheros de log
- Acceso a recursos de forma segura usando el protocolo HTTPS
- Etc.



Estructura de los mensajes HTTP de petición (método GET)

```
GET /ruta/recurso.html HTTP/1.1\r\n
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:30.0) Gecko/20100101 Firefox/30.0\r\n
Connection: keep-alive \r\n
\r\n
```



Organización de las clases

```
/** Contiene métodos estáticos para trabajar con el protocolo HTTP:
 * Obtención del método de la petición
 * Obtención del recurso solicitado
 * Obtención de las cabeceras
 * Mensajes predefinidos de respuesta
 * etc
 */
class UtilsHTTP{
}
```



Organización de las clases

```
/** Esta clase se encarga de tratar la petición de un cliente.
 * En su método run() se obtienen los flujos de E/S, usando
 * los métodos de la clase UtilsHTTP se obtiene información
 * de la petición y en función de lo que se solicita se genera
 * una respuesta
 * Finalmente se cierran los flujos y el socket
 */
class TaskStaticResponse implements Runnable{
}
```



Organización de las clases

```
/** Esta clase contiene el método main. Se encarga de
 * leer la configuración del servidor,
 * crear el ServerSocket,
 * crear un pool de hilos
 * tratar cada petición en un hilo del pool pasando una
 * nueva instancia de TaskStaticResponse
 */
class StaticHTTPServer{
}
```



StaticHTTPServer

Si queremos realizar un servidor HTTP hay que crear un `ServerSocket`, aceptar la conexión del cliente y lanzar una tarea en el pool de hilos para tratar a ese cliente.

```
Definir una serie de valores por defecto:
    Ruta de los recursos
    Prefijo para directorios privados
    Nombre del fichero con user;pass
    Puerto en el que escucha el servidor
    Número de hilos en el pool
Intentar leer los valores de un fichero de configuración
Iniciar el ServerSocket
Obtener un ExecutorService con un pool de hilos
while (true)
    Cuando un cliente se conecte tratamos la comunicación en un hilo del pool
```

TaskStaticResponse

En el código de la tarea concurrente habrá que extraer del mensaje de petición la siguiente información:

Línea de petición

```
GET /index.html HTTP/1.1 \r\n
```

Líneas con cabeceras

```
User-Agent : Mozilla ... \r\n
Date : Tue ... \r\n
\r\n
```

TaskStaticResponse I

```
En el constructor se obtienen los flujos de entrada y salida del socket

En el método run:

    Lectura de la primera línea de la petición que contiene:
        - El método (GET, POST, HEAD,...)
        - El recurso solicitado
        - La versión del protocolo

    Extracción del método HTTP de la línea leída

    Extracción de las cabeceras

    Extracción del recurso solicitado

    si el método no es null

        si el recurso es el fichero con user;pass
            escribir el mensaje de respuesta Forbidden

        sino si el método es GET y se requiere autenticación
            si el campo de cabecera Authorization es null
                escribir el mensaje de respuesta Se requiere autorización

            sino
                Obtener el usuario y password del campo de cabecera
```

TaskStaticResponse II

```
        si coinciden con el del fichero
            escribir el mensaje de respuesta con el recurso

        sino
            escribir el mensaje de respuesta Se requiere autorización

        sino si es GET
            escribir el mensaje de respuesta con el recurso
        sino si es DELETE o PUT
            escribir el mensaje de respuesta Metodo no implementado

    Si se producen excepciones
        escribir mensaje de respuesta Error interno en el servidor

    Cerrar flujos y conexión socket con el cliente
```

Mensaje de respuesta

Una vez obtenida esta información se puede buscar el recurso y construir el mensaje de respuesta:

```
200 OK HTTP/1.1\r\n
Server: Custom HTTP server\r\n
Date: Mon, 13 Oct 2014 11:02:51 GMT\r\n
Content-Length: 1458\r\n
Connection: close\r\n
\r\n
!DOCTYPE html
<html>
<head>
....
```

Mensaje de solicitud de autenticación

Si comprobamos que el recurso solicitado requiere autenticación se puede responder con:

```
HTTP/1.1 401 Unauthorized\r\n
WWW-Authenticate: Basic realm="Private resource"\r\n
Content-Length: XX\r\n
Content-Type: text/html; charset=utf-8\r\n
\r\n
<html>
<body>
<h1> Authentication is required </h1>
</body>
</html>
```

Lo que provocará que el navegador solicite usuario y contraseña y vuelva a realizar la petición incluyendo el campo de cabecera Authorization cuyo valor será la información introducida por el usuario.

Índice

- 1 *Servidor estático HTTP*
- 2 *Ejemplos de respuestas predefinidas*
- 3 *Configuración del servidor mediante un fichero de propiedades*

Credenciales incorrectas

Si comprobamos que las credenciales aportadas no son correctas:

```
HTTP/1.1 403 Forbidden\r\n
Content-Length: XX\r\n
Content-Type: text/html; charset=utf-8\r\n
Connection: close\r\n
\r\n
<html>
<body>
The action cannot be performed
</body>
</html>
```

Recurso no existe

Si el recurso solicitado no existe:

```
HTTP/1.1 404 Not Found\r\n
Content-Length: XX\r\n
Content-Type: text/html; charset=utf-8\r\n
Connection: close\r\n
\r\n
<html>
<body>
<h1> That resource is not in this server </h1>
</body>
</html>
```

Clase Properties

La clase Properties permite la lectura de pares del tipo clave/valor de un flujo.

Una vez leídas estas propiedades se pueden obtener solicitando el valor de cada clave.

Esto se puede usar para crear un fichero con valores de configuración que leerá la aplicación.

Índice

- 1 *Servidor estático HTTP*
- 2 *Ejemplos de respuestas predefinidas*
- 3 *Configuración del servidor mediante un fichero de propiedades*

Ejemplo de fichero de configuración

Ejemplo de fichero de configuración:

```
PATH=/var/web/
MAX_CLIENTS=100
PORT=8080
AUTH_PREFIX=private
CREDENTIALS=.creds
```

Lectura y obtención de las propiedades

Asumiendo que las variables (path, priv, credFile, nThreads, port) que se usan están declaradas:

```
Properties p = new Properties();
p.load(new FileInputStream("config.ini"));
path = p.getProperty("PATH", "/var/isaw/web/");
priv = p.getProperty("AUTH_PREFIX", "private");
credFile = p.getProperty("CREDENTIALS", ".creds");
nThreads = Integer.parseInt(p.getProperty("MAX_CLIENTS", "50"));
port = Integer.parseInt(p.getProperty("PORT", "8080"));
```