Presentación

El protocolo HTTP se creó a comienzos de los años 90 por investigadores del CERN. El objetivo de estos investigadores era proporcionar un protocolo simple y eficaz para la Web que en aquellos días se encontraba en sus inicios. El uso principal de este protocolo tenía que ser el intercambio de documentos de hipertexto. Desde su creación, este protocolo se ha utilizado para transferir otros elementos a parte de documentos de hipertexto.

1. Funcionamiento

El principio de funcionamiento del protocolo HTTP se basa en el tándem pregunta/respuesta. El cliente genera una pregunta con la forma de una petición HTTP. Esta petición contiene por lo menos los datos que permiten identificar el recurso solicitado por el cliente. Generalmente se le añade más información de distinta índole. La petición HTTP simplemente es un bloque de texto que transita del cliente al servidor. Este bloque de texto tiene que tener un formato concreto para que sea reconocido por el servidor. Se compone de dos partes separadas por una línea en blanco (que contiene simplemente un retorno de carro/salto de línea). La primera parte, llamada cabecera de la petición HTTP, es obligatoria. La segunda parte, llamada cuerpo de la petición HTTP, es opcional. Su presencia depende del tipo de petición HTTP. Incluso si no hay cuerpo en la petición, la línea en blanco de separación es obligatoria.

Cuando el servidor recibe la petición HTTP procedente del cliente, la analiza y realiza los tratamientos necesarios para construir la respuesta HTTP. Como sucede con las peticiones, ésta también se contruye siempre con un bloque de texto separado en dos por una línea en blanco. La primera parte, que corresponde a la cabecera de la respuesta HTTP, es obligatoria. La segunda parte, que corresponde al cuerpo de la respuesta HTTP, es opcional y depende del tipo de la respuesta HTTP.

Una respuesta HTTP solamente puede existir si se ha enviado previamente una petición HTTP al servidor. El servidor nunca toma la iniciativa de enviar datos a un cliente si éste no los ha solicitado. De hecho, es ciertamente imposible que esto sucediera debido a que de todo cliente que no le ha enviado una petición ignora simple y llanamente su existencia.

Generalmente, el protocolo TCP se utiliza para el transporte de los dos bloques de texto que forman la petición y la respuesta HTTP. Por defecto la conexión mediante este protocolo TCP se establece para cada par petición/respuesta. Sin embargo, para mejorar el uso del ancho de banda de red, la versión 1.1 del protocolo HTTP propone una solución que permite el transporte de varios pares petición/respuesta con la misma conexión TCP.

Para visualizar claramente el aspecto de un par petición/respuesta HTTP, a continuación se muestra el resultado de una captura de los datos transmitidos por la red en la generación de una petición HTTP por un navegador y la respuesta correspondiente realizada por el servidor (el cuerpo de la respuesta HTTP ha sido intencionadamente cortado en este ejemplo).

La petición HTTP relativa a la página de inicio del sitio www.eni-ecole.fr:

```
GET / HTTP/1.1
Host: www.eni-ecole.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; fr;
rv:1.9.0.15) Gecko/2009101601 Firefox/3.0.15 (.NET CLR 3.5.30729)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,en;q=0.8,fr-fr;q=0.6,en-us;q=0.4,zh;q=0.2
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

La respuesta HTTP realizada por el servidor (el cuerpo de la respuesta está truncado).

```
HTTP/1.1 200 OK
Date: Mon, 04 Jan 2010 11:20:50 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.3 with Suhosin-Patch
X-Powered-By: PHP/5.2.4-2ubuntu5.3
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
.<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
.<META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
.<META HTTP-EQUIV="EXPIRES" CONTENT="0">
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<title>ENI Ecole Informatique - Une formation, un dipl.me, un emploi</title>
<link href="eniStyle.css" rel="stylesheet" type="text/css" />
<!--[if IE 6]>
      <link rel="stylesheet" href="eniStyle6.css" type="text/css"
/>
   <![endif]-->
</head>
<body onLoad="ges_menu(document.getElementById('item7'));">
<div class="cadre">
 <div class="haut ; bleuFd">
...
...
...
</div>
</html>
```

Para demostrar que el protocolo HTTP puede usarse para transportar cualquier tipo de contenido, a continuación se muestra un par petición/respuesta que permite obtener de un servidor una imagen en formato gif.

```
HTTP/1.1 200 OK
Date: Mon, 04 Jan 2010 11:20:50 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.3 with Suhosin-
Patch
Last-Modified: Tue, 30 Sep 2008 09:45:01 GMT
ETag: "50220-1fe-45819d661c140"
Accept-Ranges: bytes
Content-Length: 510
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: image/gif

GIF89a..4.....................................................
.....................................................................
.....................................................................
.......,......4....@.pH4`...1.l:...tJ."...V..z...8.....3n.i....|N.
<.......w..............5..........-..-
.....>>....."...!...%...2........7.<<.......==6.......3..$...'...#
........??
....
,..,.....4..0.....
.Hp.....*\.P....#B.@..E..2j...... C:.Ar...(S.......
b..)....89...Sg.#.@.
.J4...H..X.....P.J.J....X.j.j5..;
```

En el caso del ejemplo, el navegador extraerá el cuerpo de la respuesta HTTP y lo tratará como una imagen en formato gif.

2. Las URL

Las URL (*Uniform Resource Locator*) están íntimamente relacionadas al protocolo HTTP. De hecho, gracias a ellas se pueden localizar los recursos que se desea recuperar mediante peticiones HTTP. Se componen de una cadena de caracteres compuesta de cinco campos. El formato que se tiene que respetar es el siguiente:

```
protocolo://identificador del servidor:número de puerto/recurso?parámetros
```

La información que contiene cada campo se describe a continuación:

- **protocolo**: indica el protocolo utilizado para acceder al recurso. En el caso del ejemplo que nos interesa, usaremos por supuesto HTTP (hay otros protocolos disponibles como ftp o mailto).

- **identificador del servidor**: esta parte de la URL permite localizar el servidor en la red. Corresponde

generalmente al FQDN (*fully qualified domain name*) del servidor. Este elemento tiene que permitir identificar sin ambigüedades el servidor en la red. Esta parte de la URL deberá transformarse en la dirección IP para que el protocolo TCP pueda establecer una conexión con el servidor. Generalmente, un servidor DNS (*Domain Name System*) se encarga de esta operación. También es posible utilizar directamente una dirección IP en una URL. Sin embargo, es más fácil de recordar un FQDN que una dirección IP (y aún lo será más cuando se utilice la versión 6 del protocolo IP).

- **número de puerto**: este dato proporciona el número de puerto TCP con el que se debe establecer la conexión. El número de puerto le sirve al protocolo TCP para identificar una aplicación particular en una máquina. De hecho, gracias a este número de puerto una máquina puede alojar varias aplicaciones, cada una de ellas usando un número de puerto diferente. El puerto 80 se asigna por defecto al protocolo HTTP. Si el servidor utiliza este número de puerto por defecto éste puede omitirse en la petición HTTP.

- **recurso**: esta parte de la URL permite determinar el recurso que se desea obtener. Puede estar compuesto por varios elementos separados por el carácter /.

- **parámetros**: cuando la petición HTTP se realiza sobre un recurso dinámico (código que se desea que ejecute el servidor) a veces es preferible poder pasar parámetros a este código. Esto es lo que permite hacer esta parte de la URL que proporciona estos datos con la forma del par nombreDeParámetro=valorDelParámetro. La aplicación tiene que estar preparada para tratar estos parámetros. Si se va a enviar más de un parámetro, hay que separar cada par correspondiente en la URL con el carácter &.

En una URL ciertos caracteres tienen un significado específico. Es el caso, por ejemplo, de los caracteres /, ?, &. Si estos caracteres tienen que usarse para otro uso que para el que han sido diseñados en una URL (en un valor de un parámetro, por ejemplo), hay que realizar una codificación de estos caracteres para asegurar la coherencia de la URL. La codificación consiste en reemplazar en la URL el carácter en cuestión por la secuencia %HH, donde HH corresponde al código ASCII del carácter expresado en hexadecimal. La tabla mostrada a continuación contiene la correspondencia de los caracteres usados más frecuentemente y su codificación asociada.

carácter	codificación
Espacio	%20 o a veces +
"	%22
%	%25
&	%26
+	%2B
.	%2E
/	%2F
?	%3F
’	%60

Esta operación de codificación se realiza automáticamente por los navegadores cuando estos construyen una URL. Por contra, si una URL tiene que construirse mediante código, será necesario realizar manualmente la codificación de los caracteres especiales. La clase URLEncoder dispone del método estático encode al que hay que proporcionar como primer parámetro la cadena de caracteres en la que se reemplazarán los caracteres especiales por su codificación. El segundo parámetro indica el juego de caracteres que se utilizará.

El código siguiente:

```
out.println(URLEncoder.encode("java&html", "UTF-8"));
```

muestra la cadena codificada:

```
java%26html
```

Ciertos navegadores o servidores relativamente antiguos imponen un límite de 255 caracteres para la longitud de la cadena de caracteres que representa la URL.