Protocolo HTTP

Introducción

El protocolo de transferencia de hipertexto (*HyperText Transfer Protocol*) es un protocolo del nivel de Aplicación usado para la transferencia de información entre sistemas, de forma clara y rápida. Este protocolo ha sido usado por el *World-Wide Web* desde 1990.

Este protocolo permite usar una serie de métodos para indicar la finalidad de la petición. Se basa en otros conceptos y estándares como *Uniform Resource Identifier* (URI), *Uniform Resource Location* (URL) y *Uniform Resource Name* (URN), para indicar el recurso al que hace referencia la petición. Los mensajes se pasan con un formato similar al usado por el *Internet Mail* y el *Multipurpose Internet Mail Extensions* (MIME).

El protocolo HTTP se basa en un paradigma de **peticiones** y **respuestas**. Un cliente envía una petición en forma de método, una URI, y una versión de protocolo seguida de los modificadores de la petición de forma parecida a un mensaje MIME, información sobre el cliente y al final un posible contenido. El servidor contesta con una línea de estado que incluye la versión del protocolo y un código que indica éxito o error, seguido de la información del servidor en forma de mensaje MIME y un posible contenido.

Generalmente es el cliente el que inicia la comunicación HTTP y consiste en la petición de un recurso del servidor. Puede hacerse de forma directa al servidor o a través de intermediarios.

Se han utilizado los protocolos HTTP/0.9, HTTP/1.0 y HTTP/1.1

Sintaxis de la petición

> El esquema "*http*" se usa para localizar recursos en la red por medio del protocolo http.

La sintaxis de la petición es la siguiente:

`http://dirección:puerto/[path]`

Donde **`dirección`** es el nombre de un dominio de Internet o una dirección IP, el **`puerto`** es un número que indica el puerto al que se envía la petición y el **`path`** indica la ruta al recurso al que se accede.

Si no se indica un número de puerto, por defecto se supone que se accede al puerto 80. Si no se indica un *path*, entonces se supone que este es la raíz del filesystem "**`/`**"'.

Mensaje HTTP

Un mensaje http consiste en una petición de un cliente al servidor y en la respuesta del servidor al cliente.

Las peticiones y respuestas pueden ser simples o completas. La diferencia es que en las peticiones y respuestas completas se envían cabeceras y un contenido. Este contenido se

pone después de las cabeceras dejando una línea vacía entre las cabeceras y el contenido. En el caso de peticiones simples, sólo se puede usar el método **GET** y no hay contenido. Si se trata de una respuesta simple, entonces ésta sólo consta de contenido. Esta diferenciación entre simples y completas se tiene para que el protocolo HTTP/1.0 pueda atender peticiones y enviar respuestas del protocolo HTTP/0.9.

Petición

Una petición de un cliente a un servidor ha de incluir el método que se aplica al recurso, el identificador del recurso y la versión del protocolo que usa para realizar la petición.

Para mantener la compatibilidad con el protocolo HTTP/0.9 se permite una petición simple con el formato:

GET SP URI CRLF

Donde ***SP*** es un espacio, ***URI*** es la URI del recurso al que hace referencia la petición y ***CRLF*** es un retorno de carro y nueva línea.

En el caso de que la petición se haga con el protocolo HTTP/1.0 o con el protocolo HTTP/1.1 la petición sigue el formato:

Línea de petición

***(Cabeceras)**

CRLF

[Contenido]

La línea de petición comienza indicando el método, seguido de la URI de la petición y la versión del protocolo, finalizando la línea con CRLF:

Método SP URI de la petición SP Versión del protocolo CRLF

En el caso del protocolo HTTP/0.9 sólo se permite el método GET, con el protocolo HTTP/1.0 GET, POST y HEAD y con el protocolo HTTP/1.1 OPTIONS, GET, HEAD, POST, PUT, DELETE y TRACE. En caso de que un servidor tenga implementado un método, pero no está permitido para el recurso que se pide, entonces ha de devolver un código de estado 405 (método no permitido). Si lo que ocurre es que no tiene implementado el método, entonces devuelve un código 501 (no implementado). Los únicos métodos que deben soportar los servidores de forma obligatoria son los métodos GET y HEAD.

En el apartado de cabeceras, éstas pueden ser de tres tipos: cabeceras generales, de petición y de entidad. Las cabeceras generales son las que se aplican tanto a peticiones como a respuestas, pero no al contenido que se transmite. Las cabeceras de petición permiten al cliente pasar información al servidor sobre la petición y sobre el cliente. Las cabeceras de entidad permiten definir información adicional sobre el contenido que se transmite y en caso de que no haya contenido, sobre el recurso al que se quiere acceder con la petición.

El contenido (si está presente) está en un formato con una codificación definida en las cabeceras de entidad.

Respuesta

Después de recibir e interpretar una petición, el servidor debe responder con un mensaje HTTP. Este mensaje tiene el siguiente formato:

```
Línea de estado
*( Cabeceras )
CRLF
[ Contenido ]
```

La **línea de estado** es la primera línea de la respuesta y consiste en la versión de protocolo que se utiliza, seguida de una indicación de estado numérica a la que puede ir asociada una frase explicativa. El formato es el siguiente:

```
Versión del protocolo SP Código de estado SP Frase explicativa CRLF
```

El **código de estado** es un número de 3 dígitos que indica si la petición ha sido atendida satisfactoriamente o no, y en caso de no haber sido atendida, indica la causa. Los códigos se dividen en cinco clases definidas por el primer dígito del código de estado. Así tenemos:

- **1xx**: **Informativo**. La petición se recibe y sigue el proceso. Esta familia de respuestas indican una respuesta provisional. Este tipo de respuesta está formada por la línea de estado y las cabeceras. Un servidor envía este tipo de respuesta en casos experimentales.
- **2xx**: **Éxito**. La acción requerida por la petición ha sido recibida, entendida y aceptada.
- **3xx**: **Redirección**. Para completar la petición se han de tomar más acciones.
- **4xx**: **Error del cliente**. La petición no es sintácticamente correcta y no se puede llevar a cabo.
- **5xx**: **Error del servidor.** El servidor falla al atender la petición que aparentemente es correcta.

Algunos de los códigos más comúnmente usados y las frases asociadas son:

- 100, continuar.
- 101, cambio de protocolo.
- 200, éxito.
- 201, creado.
- 202, aceptado.
- 203, información no autoritativa.
- 204, sin contenido.
- 205, contenido reestablecido.
- 206, contenido parcial.
- 300, múltiples elecciones.
- 301, movido permanentemente.
- 302, movido temporalmente.
- 303, ver otros.
- 304, no modificado.
- 305, usar *proxy*.
- 400, petición errónea.
- 401, no autorizado.
- 402, pago requerido.
- 403, prohibido.
- 404, no encontrado.
- 405, método no permitido.

- 406, no se puede aceptar.
- 407, se requiere autentificación *proxy*.
- 408, límite de tiempo de la petición.
- 409, conflicto.
- 410, *gone*.
- 411, tamaño requerido.
- 412, falla una precondición.
- 413, contenido de la petición muy largo.
- 414, URI de la petición muy largo.
- 415, campo *media type* requerido.
- 500, error interno del servidor.
- 501, no implementado.
- 502, puerta de enlace errónea.
- 503, servicio no disponible.
- 504, tiempo límite de la puerta de enlace.
- 505, versión de protocolo HTTP no soportada.

La frase explicativa, es eso, una frase corta que explica el código de estado enviado al cliente.

Se pueden usar más códigos, pero las aplicaciones HTTP no tienen que conocer todos los códigos definidos y su significado, pero sí están obligadas a conocer su clase y tratar los códigos desconocidos como el primer código de la clase (x00).

En cuanto a las cabeceras de la respuesta, son de tres tipos: las cabeceras generales, las cabeceras de la respuesta y las cabeceras de entidad.

Las cabeceras de respuesta permiten al servidor enviar información adicional al cliente sobre la respuesta. Estos campos dan información sobre el servidor y acceso al recurso pedido.

Métodos

Un método se dice que es seguro si no provocan ninguna otra acción que no sea la de devolver algo (no produce efectos laterales). Estos métodos son el método GET y el método HEAD. Para realizar acciones inseguras (las que afectan a otras acciones) se pueden usar los métodos POST, PUT y DELETE. Aunque esto está definido así, no se puede asegurar que un método seguro no produzca efectos laterales, porque depende de la implementación del servidor.

Un método es idempotente si los efectos laterales para N peticiones son los mismos que para una sola petición. Los métodos idempotentes son los métodos GET, HEAD, PUT y DELETE.

Método OPTIONS

Este método representa un petición de información sobre las opciones de comunicación disponibles en la cadena petición-respuesta identificada por la URI de la petición. Esto permite al cliente conocer las opciones y requisitos asociados con un recurso o las capacidades del servidor.

La respuesta sólo debe incluir información sobre las opciones de comunicación.

Si la URI es "*"', entonces la petición se aplica al servidor como un conjunto. Es decir, contesta características opcionales definidas por el servidor, extensiones del protocolo, ...

Método GET

El método GET requiere la devolución de información al cliente identificada por la URI. Si la URI se refiere a un proceso que produce información, se devuelve la información y no la fuente del proceso.

El método GET pasa a ser un GET condicional si la petición incluye las cabeceras **If-Modified-Since**, **If-Unmodified-Since**, **If-Match**, **If-None-Match** o **If-Range**. Estas cabeceras hacen que el contenido de la respuesta se transmita sólo si se cumplen unas condiciones determinadas por esas cabeceras. Esto se hizo para reducir el tráfico en las redes.

También hay un método GET parcial, con el que se envía sólo parte del contenido del recurso requerido. Esto ocurre cuando la petición tiene una cabecera *Range*. Al igual que el método GET condicional, el método GET parcial se creó para reducir el tráfico en la red.

Método HEAD

El método HEAD es igual que el método GET, salvo que el servidor no tiene que devolver el contenido, sólo las cabeceras. Estas cabeceras que se devuelven en el método HEAD deberían ser las mismas que las que se devolverían si fuese una petición GET.

Este método se puede usar para obtener información sobre el contenido que se va a devolver en respuesta a la petición. Se suele usar también para chequear la validez de *links*, accesibilidad y modificaciones recientes.

Método POST

El método POST se usa para hacer peticiones en las que el servidor destino acepta el contenido de la petición como un nuevo subordinado del recurso pedido. El método POST se creó para cubrir funciones como la de enviar un mensaje a grupos de usuarios, dar un bloque de datos como resultado de un formulario a un proceso de datos, añadir nuevos datos a una base de datos, ...

La función llevada a cabo por el método POST está determinada por el servidor y suele depender de la URI de la petición. El resultado de la acción realizada por el método POST puede ser un recurso que no sea identificable mediante una URI.

Método PUT

El método PUT permite guardar el contenido de la petición en el servidor bajo la URI de la petición. Si esta URI ya existe, entonces el servidor considera que esta petición proporciona una versión actualizada del recurso. Si la URI indicada no existe y es válida para definir un nuevo recurso, el servidor puede crear el recurso con esa URI. Si se crea un nuevo recurso, debe responder con un código 201 (creado), si se modifica se contesta con un código 200 (OK) o 204 (sin contenido). En caso de que no se pueda crear el recurso se devuelve un mensaje con el código de error apropiado.

La principal diferencia entre POST y PUT se encuentra en el significado de la URI. En el caso del método POST, la URI identifica el recurso que va a manejar en contenido, mientras que en el PUT identifica el contenido.

Un recurso puede tener distintas URI.

Método DELETE

Este método se usa para que el servidor borre el recurso indicado por la URI de la petición. No se garantiza al cliente que la operación se lleve a cabo aunque la respuesta sea satisfactoria.

Método TRACE

Este método se usa para saber si existe el receptor del mensaje y usar la información para hacer un diagnóstico. En las cabeceras el campo *Via* sirve para obtener la ruta que sigue el mensaje. Mediante el campo ***Max-Forwards*** se limita el número de pasos intermedios que puede tomar. Esto es útil para evitar bucles entre los ***proxy***.

La petición con el método TRACE no tiene contenido.

Cabeceras generales

Los campos de este tipo de cabeceras se aplican tanto a las peticiones como a las respuestas, pero no al contenido de los mensajes.

Estas cabeceras son:

- **Cache-Control**, son directivas que se han de tener en cuenta a la hora de mantener el contenido en una caché.
- **Connection**, permite especificar opciones requeridas para una conexión.
- **Date**, representa la fecha y la hora a la que se creó el mensaje.
- **Pragma**, usado para incluir directivas de implementación.
- **Transfer-Encoding**, indica la codificación aplicada al contenido.
- **Upgrade**, permite al cliente especificar protocolos que soporta.
- **Via**, usado por pasarelas y *proxies* para indicar los pasos seguidos.

Cabeceras de petición

Este tipo de cabeceras permite al cliente pasar información adicional al servidor sobre la petición y el propio cliente.

Estas cabeceras son las siguientes:

- **Accept**, indican el tipo de respuesta que acepta.
- **Accept-Charset**, indica los conjuntos de caracteres que acepta.
- **Accept-Encoding**, que tipo de codificación acepta.
- **Accept-Language**, tipo de lenguaje de la respuesta que se prefiere.
- **Authorization**, el agente de usuario quiere autentificarse con el servidor.
- **From**, contiene la dirección de correo que controla en agente de usuario.
- **Host**, especifica la máquina y el puerto del recurso pedido.
- **If-Modified-Since**, para el GET condicional.
- **If-Match**, para el GET condicional.
- **If-None-Match**, para el GET condicional.
- **If-Range**, para el GET condicional.
- **If-Unmodified-Since**, para el GET condicional.

- **Max-Forwards**, indica el máximo número de elementos por los que pasa.
- **Proxy-Authorization**, permite que el cliente se identifique a un *proxy*.
- **Range**, establece un rango de *bytes* del contenido.
- **Referer**, indica la dirección donde obtuvo la URI de la petición.
- **User-Agent**, información sobre el agente que genera la petición.

Cabeceras de respuesta

Permiten al servidor pasar información adicional al cliente sobre la respuesta, el propio servidor y el recurso solicitado.

Son los campos:

- **Age**, estimación del tiempo transcurrido desde que se creó la respuesta.
- **Location**, se usa par a redirigir la petición a otra URI.
- **Proxy-Authenticate**, ante una respuesta con el código 407 (autenticación *proxy* requerida), indica el esquema de autenticación.
- **Public**, da la lista de métodos soportados por el servidor.
- **Retry-After**, ante un servicio no disponible da una fecha para volver a intentarlo.
- **Server**, información sobre el servidor que maneja las peticiones.
- **Vary**, indica que hay varias respuestas y el servidor ha escogido una.
- **Warning**, usada para aportar información adicional sobre el estado de la respuesta.
- **WWW-Authenticate**, indica el esquema de autenticación y los parámetros aplicables a la URI.

Cabeceras de entidad

Como su nombre indica, los campos de este tipo aportan información sobre el contenido del mensaje o si no hay contenido, sobre el recurso al que hace referencia la URI de la petición.

Los campos de este tipo son:

- **Allow**, da los métodos soportados por el recurso designado por la URI.
- **Content-Base**, indica la URI base para resolver las URI relativas.
- **Content-Encoding**, indica una codificación adicional aplicada al contenido (a parte de la aplicada por el tipo).
- **Content-Language**, describe el idioma del contenido.
- **Content-Length**, indica el tamaño del contenido del mensaje.
- **Content-Location**, da información sobre la localización del recurso que da el contenido del mensaje.
- **Content-MD5**, es un resumen en formato MD5 (RFC 1864) para chequear la integridad del contenido.
- **Content-Range**, en un GET parcial, indica la posición del contenido.
- **Content-Type**, indica el tipo de contenido que es.
- **Etag**, define una marca para el contenido asociado.
- **Expires**, indica la fecha a partir de la cual la respuesta deja de ser válida.
- **Last-Modified**, indica la fecha de la última modificación.

Cookies

Las *cookies* se han descrito para conseguir sesiones HTTP con un estado asociado. Por definición, las distintas peticiones HTTP son independientes entre sí, es decir, el servidor responde a la petición sin tener en cuenta las peticiones anteriores del cliente. Mediante las *cookies* se puede intercambiar información de estado que afecta a una serie de peticiones y respuestas, es decir, una sesión.

Hay varios contextos en los que se puede establecer una sesión:

- Cada sesión tiene un principio y un final.
- Las sesiones son cortas.
- El agente de usuario o el servidor terminan la sesión.
- La sesión está implícita en el intercambio de información de estado.

Funcionamiento

Para hacer operativas las *cookies* se han definido dos cabeceras, **Set-Cookie** y **Cookie**. Con estas cabeceras se puede establecer una nueva *cookie* e informar de las *cookies* existentes.

El servidor

Es el servidor el que inicia la sesión al responder al cliente con un mensaje que tiene una cabecera para establecer una *cookie* ("***Set-Cookie***''). Cuando el cliente recibe esta respuesta, en la siguiente petición incorporará una cabecera para informar de las *cookies* que tiene ("***Cookie***''). Ante esta petición puede tener en cuenta la *cookie* o no para dar la respuesta, y para ello puede establecer el mismo valor u otro valor para la *cookie* o no enviar ninguna *cookie*. Para acabar la sesión, el servidor debe establecer una *cookie* con duración 0.

El servidor puede tener varias cabeceras para establecer varias *cookies* en una misma respuesta, aunque si el mensaje pasa por un *proxy* o una pasarela, pueden convertirlo en una sola cabecera para establecer *cookies*.

Para establecer una *cookie* se usa la cabecera **Set-Cookie**, cuya sintaxis es:

“Set-Cookie:” nombre “=” valor *(“;” modificador)

Con esto se establece que por lo menos ha de haber un par nombre, valor y cada par puede tener o no modificadores.

Los modificadores de un par nombre, valor pueden ser:

- **“Comment” “=” valor**, el servidor informa del uso de la *cookie*.
- **“Domain” “=” valor**, indica el dominio en el que la *cookie* es válida.
- **“Max-Age” “=” valor**, validez de la *cookie* en segundos, después se descarta.
- **“Path” “=” valor**, indica el subconjunto de URL a las que afecta la *cookie*.
- **“Secure”**, indica que el cliente debe usar en entornos seguros para contactar con el servidor.

- **“Versión” “=” valor**, la versión de la especificación de mantenimiento de estado que es.

Se pueden enviar varias *cookies* en una misma cabecera separando los pares nombre, valor con “,''.

Además de enviar la *cookie*, el servidor puede establecer las condiciones para que la *cookie* se ponga en una caché o no.

El cliente

El cliente trata por separado las distintas informaciones de estado que le llegan por medio de respuestas que establecen *cookies*. Además toma valores por defecto para los atributos que no están presentes de los pares nombre, valor.

Además el cliente puede rechazar una *cookie* por razones de seguridad o violaciones de la privacidad. Para rechazar una *cookie* tiene una serie de reglas que aplica en cada caso.

Si el cliente recibe una *cookie* con un nombre que ya tiene asociado a otra *cookie* y cuyos atributos **Domain** y **Path** son exactamente iguales, entonces la nueva *cookie* reemplaza a la anterior. Además si la nueva *cookie* tiene el atributo **Max-Age** con un valor de 0, entonces elimina la *cookie* en vez de reemplazarla.

Como el cliente tiene un espacio limitado para las *cookies*, ha de ir eliminando *cookies* antiguas para hacer sitio a las nuevas. Esta operación la puede realizar siguiendo algún algoritmo como el **LRU** (*Least Recently Used*), o **FIFO** (*First In First Out*).

Si una *cookie* tiene el atributo **Comment**, entonces el cliente ha de almacenar el comentario para que el usuario (humano) pueda leerlo.

Cuando el cliente realiza una petición al servidor, lo hace por medio de una URI. Para cada petición, además del mensaje envía una cabecera “**Cookie**'' para informar al servidor de las *cookies* que tiene y que afectan a esa petición. La sintaxis para esta cabecera es la siguiente:

“Cookie:” versión *((“;” | “,”) nombre “=” valor [";" path] [";" dominio])

Donde **versión** es:

“$Versión” “=” valor

path es:

“$Path” “=” valor

Y **dominio** es:

“$Domain” “=” valor

El valor de la versión es el del atributo v**ersión** si alguna de las *cookies* lo impone, si no es 0. El valor de **path** ha de ser el impuesto por las *cookies* si alguna lo establece, si no se omite. El caso del **dominio** se trata igual que el del *path*.

En la lista de *cookies* aparecen las que satisfacen los criterios:

- El nombre del servidor ha de tener el mismo dominio que la *cookie*.
- El *path* de la *cookie* ha de ser un prefijo de la URI que representa la petición.
- El límite de tiempo de la *cookie* no ha sido sobrepasado.

Cuando el servidor recibe una petición con una cabecera *cookie*, ha de interpretar los pares nombre, valor teniendo en cuenta que los nombres que comienzan por el símbolo "$" son especiales y hacen referencia a la *cookie* anterior, y si no hay una *cookie* anterior, hacen referencia a todas las *cookies* de esta cabecera.

Compatibilidad con el *Netscape*

- *Netscape* hizo una propuesta original diferente sobre las *cookies*, aunque es muy parecida a la explicada.
- Una diferencia está en que en vez de usar el campo **`Max-Age`** usa en su lugar el campo **`Expires`**.
- Además en los pares nombre, valor de las *cookies* no acepta el símbolo "."
- Por último, en la propuesta original de *Netscape* no se permitían espacios en blanco entre el símbolo "=" y el nombre o el valor.

Abreviaturas utilizadas

Las abreviaturas utilizadas son las siguientes:

- API, *Aplication Programming Interface*
- ASP, *Active Server Pages*
- CGI, *Common Gateway Interface*
- ESI, *Erlang Scripting Interface*
- HTML, *HyperText Markup Language*
- HTTP, *HyperText Transfer Protocol*
- HTTPS, HTTP sobre SSL
- JSP, *Java Server Pages*
- MIME, *Multipurpose Internet Mail Extensions*
- NSAPI, *Netscape Server* API
- PHP, *Hypertext Preprocessor*
- SSL, *Secure Socket Layer*
- URI, *Uniform Resource Identifier*
- URL, *Uniform Resource Location*