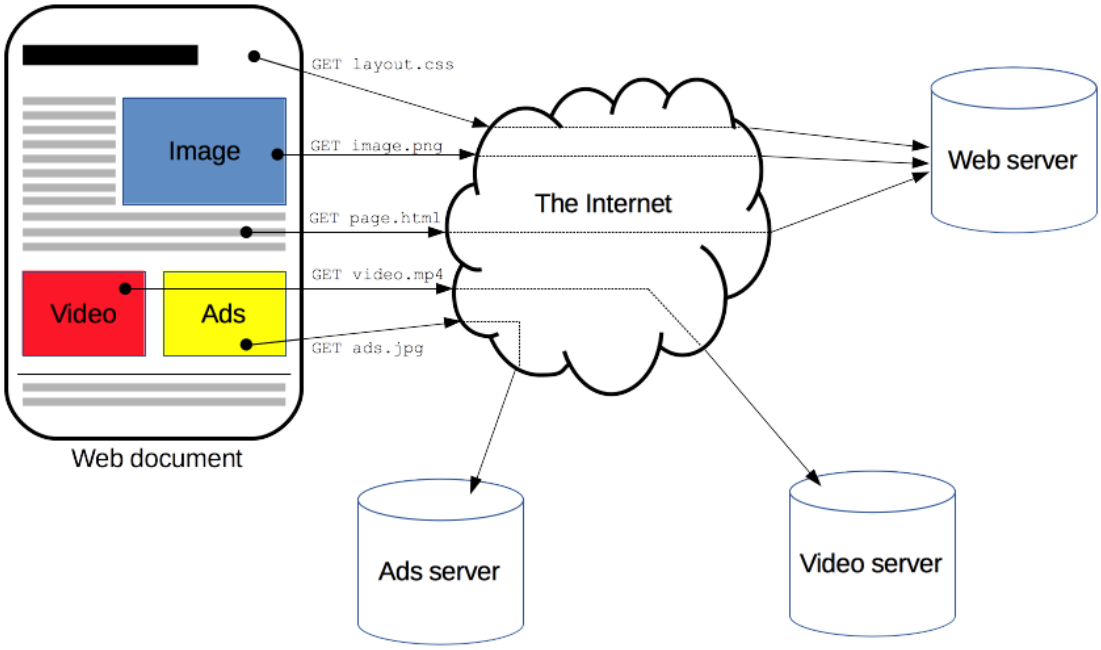
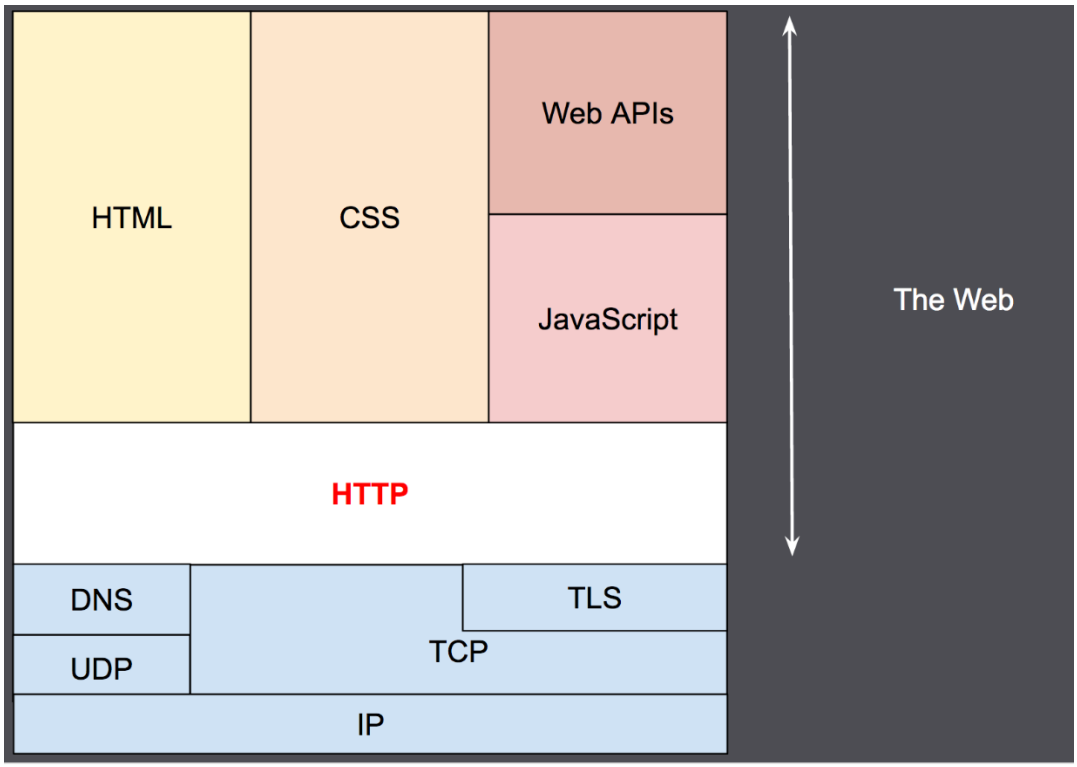
Generalidades del protocolo HTTP

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts, etc...



Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman *peticiones*, y los mensajes enviados por el servidor se llaman *respuestas*.

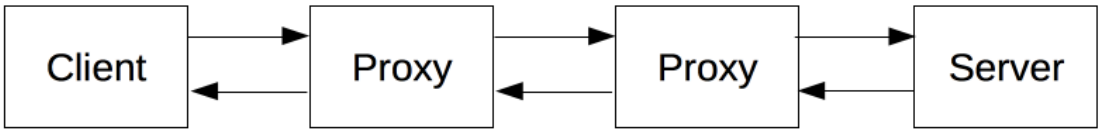


Diseñado a principios de la década de 1990, HTTP es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación, y se transmite sobre el protocolo TCP, o el protocolo encriptado TLS (en-US), aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), sino que, además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto.

Arquitectura de los sistemas basados en HTTP

HTTP es un protocolo basado en el principio de cliente-servidor: las peticiones son enviadas por una entidad: el agente del usuario (o un proxy a petición de uno). La mayoría de las veces el agente del usuario (cliente) es un navegador Web, pero podría ser cualquier otro programa, como por ejemplo un programa-robot, que explore la Web, para adquirir datos de su estructura y contenido para uso de un buscador de Internet.

Cada petición individual se envía a un servidor, el cual la gestiona y responde. Entre cada *petición* y *respuesta*, hay varios intermediarios, normalmente denominados proxies (en-US), los cuales realizan distintas funciones, como: gateways o caches.



En realidad, hay más elementos intermedios, entre un navegador y el servidor que gestiona su petición: hay otros tipos de dispositivos: como *routers*, *modems* ... Es gracias a la arquitectura en capas de la Web, que estos intermediarios, son transparentes al navegador y al servidor, ya que HTTP se apoya en los protocolos de red y transporte. HTTP es un protocolo de aplicación, y por tanto se apoya sobre los anteriores. Aunque para diagnosticar problemas en redes de comunicación, las capas inferiores son irrelevantes para la definición del protocolo HTTP .

Cliente: el agente del usuario

El agente del usuario es cualquier herramienta que actué en representación del usuario. Esta función es realizada en la mayor parte de los casos por un navegador Web. Hay excepciones, como el caso de programas específicamente usados por desarrolladores para desarrollar y depurar sus aplicaciones.

El navegador es **siempre** el que inicia una comunicación (petición), y el servidor nunca la comienza (hay algunos mecanismos que permiten esto, pero no son muy habituales).

Para poder mostrar una página Web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts, hojas de estilo (CSS), y otros datos que necesite (normalmente vídeos y/o imágenes). El navegador, une todos estos documentos y datos, y compone el resultado final: la página Web. Los scripts, los ejecuta también el navegador, y también pueden generar más peticiones de datos en el tiempo, y el navegador, gestionará y actualizará la página Web en consecuencia.

Una página Web, es un documento de hipertexto (HTTP), luego habrá partes del texto en la página que puedan ser enlaces (links) que pueden ser activados (normalmente al hacer click sobre ellos) para hacer una petición de una nueva página Web, permitiendo así dirigir su agente de usuario y navegar por la Web. El navegador, traduce esas direcciones en peticiones de HTTP, e interpretará y procesará las respuestas HTTP, para presentar al usuario la página Web que desea.

El servidor Web

Al otro lado del canal de comunicación, está el servidor, el cual *"sirve"* los datos que ha pedido el cliente. Un servidor conceptualmente es una unica entidad, aunque puede estar formado por varios elementos, que se reparten la carga de peticiones, (load balancing), u otros programas, que gestionan otros

computadores (como cache, bases de datos, servidores de correo electrónico, ...), y que generan parte o todo el documento que ha sido pedido.

Un servidor no tiene que ser necesariamente un único equipo físico, aunque sí que varios servidores pueden estar funcionando en un único computador. En el estándar HTTP/1.1 y `Host` , pueden incluso compartir la misma dirección de IP.

Proxies

Entre el cliente y el servidor, además existen distintos dispositivos que gestionan los mensajes HTTP. Dada la arquitectura en capas de la Web, la mayoria de estos dispositivos solamente gestionan estos mensajes en los niveles de protocolo inferiores: capa de transporte, capa de red o capa física, siendo así transparentes para la capa de comunicaciones de aplicación del HTTP, además esto aumenta el rendimiento de la comunicación. Aquellos dispositivos, que sí operan procesando la capa de aplicación son conocidos como proxies. Estos pueden ser transparentes, o no (modificando las peticiones que pasan por ellos), y realizan varias funciones:

- caching (la caché puede ser pública o privada, como la caché de un navegador)
- filtrado (como un anti-virus, control parental, ...)
- balanceo de carga de peticiones (para permitir a varios servidores responder a la carga total de peticiones que reciben)
- autentificación (para el control al acceso de recursos y datos)
- registro de eventos (para tener un histórico de los eventos que se producen)

Características clave del protocolo HTTP

HTTP es sencillo

Incluso con el incremento de complejidad, que se produjo en el desarrollo de la versión del protocolo HTTP/2, en la que se encapsularon los mensajes, HTTP está pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más facil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.

HTTP es extensible

Presentadas en la versión HTTP/1.0, las cabeceras de HTTP han hecho que este protocolo sea fácil de ampliar y de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.

HTTP es un protocolo con sesiones, pero sin estados

HTTP es un protocolo sin estado, es decir: no guarda ningún dato entre dos peticiones en la misma sesión. Esto crea problemáticas, en caso de que los usuarios requieran interactuar con determinadas páginas Web de forma ordenada y coherente, por ejemplo, para el uso de "cestas de la compra" en páginas que utilizan en comercio electrónico. Pero, mientras HTTP ciertamente es un protocolo sin estado, el uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. Usando la capacidad de ampliación del protocolo HTTP, las cookies permiten crear un contexto común para cada sesión de comunicación.

HTTP y conexiones

Una conexión se gestiona al nivel de la capa de trasporte, y por tanto queda fuera del alcance del protocolo HTTP. Aún con este factor, HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error). De los dos protocolos más comunes en Internet, TCP es fiable, mientras que UDP, no lo es. Por lo tanto, HTTP, se apoya en el uso del protocolo TCP, que está orientado a conexión, aunque una conexión continua no es necesaria siempre.

En la versión del protocolo HTTP/1.0, habría una conexión TCP por cada petición/respuesta intercambiada, presentando esto dos grandes inconvenientes: abrir y crear una conexión requiere varias rondas de mensajes y por lo tanto resultaba lento. Esto sería más eficiente si se mandaran varios mensajes.

Para atenuar estos inconvenientes, la versión del protocolo HTTP/1.1 presentó el 'pipelining' y las conexiones persistentes: el protocolo TCP que lo transmitía en la capa inferior se podía controlar parcialmente, mediante la cabecera 'Connection'. La versión del protocolo HTTP/2 fue más allá y usa multiplexación de mensajes sobre una única conexión, siendo así una comunicación más eficiente.

Todavía hoy se sigue investigando y desarrollando para conseguir un protocolo de transporte más conveniente para el HTTP. Por ejemplo, Google está experimentado con QUIC, que se apoya en el protocolo UDP y presenta mejoras en la fiabilidad y eficiencia de la comunicación.

¿Qué se puede controlar con HTTP?

La característica del protocolo HTTP de ser ampliable, ha permitido que durante su desarrollo se hayan implementado más funciones de control y funcionalidad sobre la Web: caché o métodos de identificación o autentificación fueron temas que se abordaron pronto en su historia. Al contrario, la relajación de la restricción de origen solo se ha abordado en los años de la década de 2010.

Se presenta a continuación una lista con los elementos que se pueden controlar con el protocolo HTTP:

- *Cache*
  El cómo se almacenan los documentos en la caché, puede ser especificado por HTTP. El servidor puede indicar a los proxies y clientes, que quiere almacenar y durante cuánto tiempo. Aunque el cliente, también puede indicar a los proxies de caché intermedios que ignoren el documento almacenado.
- *Flexibilidad del requisito de origen*
  Para prevenir invasiones de la privacidad de los usuarios, los navegadores Web, solamente permiten a páginas del mismo origen, compartir la información o datos. Esto es una complicación para el servidor, asi que mediante cabeceras HTTP, se puede flexibilizar o relajar esta división entre cliente y servidor
- *Autentificación*
  Hay páginas Web, que pueden estar protegidas, de manera que solo los usuarios autorizados puedan acceder. HTTP provee de servicios básicos de autentificación, por ejemplo, mediante el uso de cabeceras como: `WWW-Authenticate`, o estableciendo una sesión especifica mediante el uso de HTTP cookies.
- *Proxies y tunneling*
  Servidores y/o clientes pueden estar en intranets y esconder así su verdadera dirección IP a otros. Las peticiones HTTP utilizan los proxies para acceder a ellos. Pero no todos los proxies son HTTP proxies. El protocolo SOCKS, por ejemplo, opera a un nivel más bajo. Otros protocolos, como el FTP, pueden ser servidos mediante estos proxies.
- *Sesiones*
  El uso de HTTP cookies permite relacionar peticiones con el estado del servidor. Esto define las sesiones, a pesar de que por definición el protocolo HTTP es un protocolo sin estado. Esto es muy útil no sólo para aplicaciones de comercio electrónico, sino también para cualquier sitio que permita configuración al usuario.

Flujo de HTTP

Cuando el cliente quiere comunicarse con el servidor, tanto si es directamente con él, o a través de un proxy intermedio, realiza los siguientes pasos:

1. Abre una conexión TCP: la conexión TCP se usará para hacer una petición, o varias, y recibir la respuesta. El cliente pude abrir una conexión nueva, reusar una existente, o abrir varias a la vez hacia el servidor.
2. Hacer una petición HTTP: Los mensajes HTTP (previos a HTTP/2) son legibles en texto plano. A partir de la versión del protocolo HTTP/2, los mensajes se encapsulan en franjas, haciendo que no sean directamente interpretables, aunque el principio de operación es el mismo.

```
GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr
```

3. Leer la respuesta enviada por el servidor:

```
HTTP/1.1 200 OK

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

ETag: "51142bc1-7449-479b075b2891b"

Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html



<!DOCTYPE html... (here comes the 29769 bytes of the requested
web page)
```

4. Cierre o reuso de la conexión para futuras peticiones.

Si está activado el HTTP *pipelining*, varias peticiones pueden enviarse sin tener que esperar que la primera respuesta haya sido satisfecha. Este procedimiento es difícil de implementar en las redes de computadores actuales, donde se mezclan software antiguos y modernos. Así que el HTTP *pipelining* ha sido substituido en HTTP/2 por el multiplexado de varias peticiones en una sola trama

Mensajes HTTP

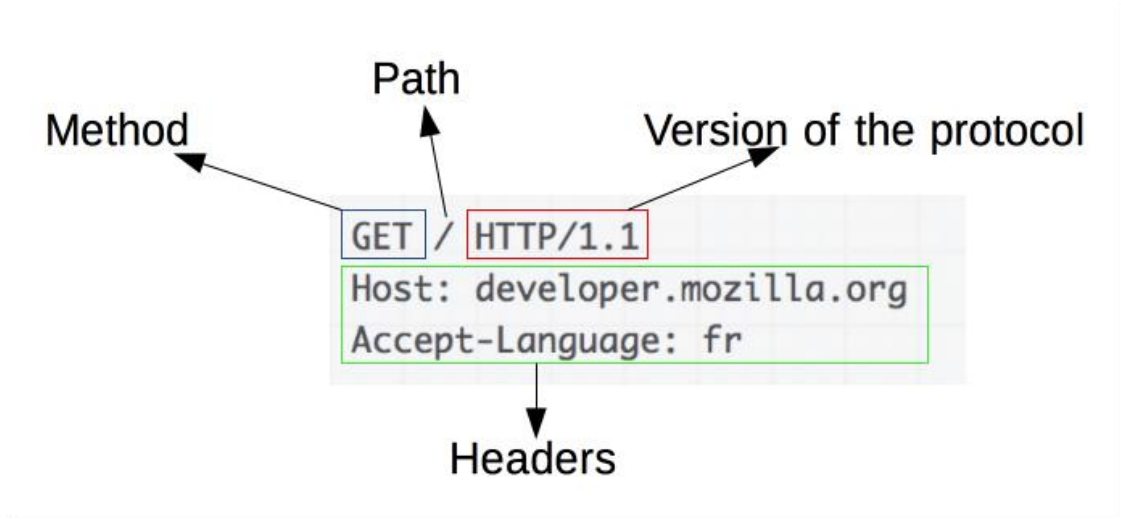
En las versiones del protocolo HTTP/1.1 y anteriores los mensajes eran de formato texto y eran totalmente comprensibles directamente por una persona. En HTTP/2, los mensajes están estructurados en un nuevo formato binario y las

tramas permiten la compresión de las cabeceras y su multiplexación. Así pues, incluso si solamente parte del mensaje original en HTTP se envía en este formato, la semántica de cada mensaje es la misma y el cliente puede formar el mensaje original en HTTP/1.1. Luego, es posible interpretar los mensajes HTTP/2 en el formato de HTTP/1.1.

Existen dos tipos de mensajes HTTP: peticiones y respuestas, cada uno sigue su propio formato.

Peticiones

Un ejemplo de petición HTTP:

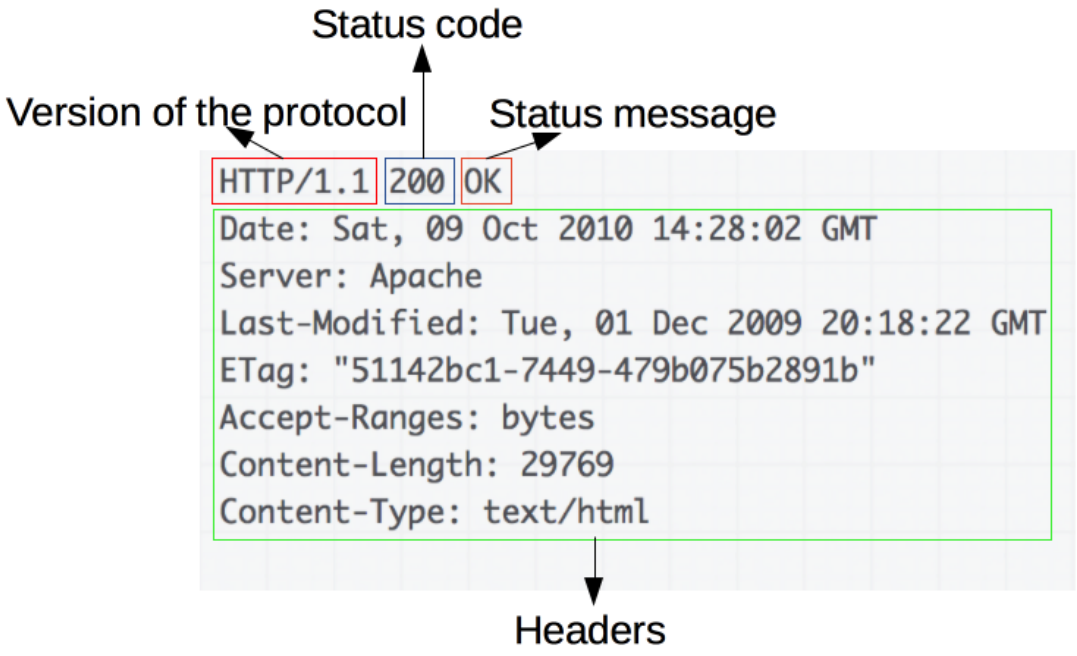


Una petición de HTTP está formada por los siguientes campos:

- Un **método** HTTP, normalmente pueden ser un verbo, como: `GET`, `POST` o un nombre como: OPTIONS (en-US) o HEAD (en-US), que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.
- La dirección del recurso pedido; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (`http://`), el dominio (aquí `developer.mozilla.org`), o el puerto TCP (aquí el 80).
- La versión del protocolo HTTP.
- Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
- O un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.

Respuestas

Un ejemplo de repuesta:



Las respuestas están formadas por los siguientes campos:

- La versión del protocolo HTTP que están usando.
- Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
- Un mensaje de estado, una breve descripción del código de estado.
- Cabeceras HTTP, como las de las peticiones.
- Opcionalmente, el recurso que se ha pedido.

Conclusión

El protocolo HTTP es un protocolo ampliable y fácil de usar. Su estructura cliente-servidor, junto con la capacidad para usar cabeceras, permite a este protocolo evolucionar con las nuevas y futuras aplicaciones en Internet.

Aunque la versión del protocolo HTTP/2 añade algo de complejidad, al utilizar un formato en binario, esto aumenta su rendimiento, y la estructura y semántica de los mensajes es la misma desde la versión HTTP/1.0.

Evolución del protocolo HTTP

HTTP es el protocolo en el que se basa la Web. Fue inventado por Tim Berners-Lee entre los años 1989-1991, HTTP ha visto muchos cambios, manteniendo la mayor parte de su simplicidad y desarrollando su flexibilidad. HTTP ha evolucionado, desde un protocolo destinado al intercambio de archivos en un entorno de un laboratorio semi-seguro, al actual laberinto de Internet, sirviendo ahora para el intercambio de imágenes, vídeos en alta resolución y en 3D.

Invención de la World Wide Web

En 1989, mientras trabajaba en el CERN, Tim Berners-Lee escribió una propuesta para desarrollar un sistema de hipertexto sobre Internet. Inicialmente lo llamó: '*Mesh'* (malla, en inglés), y posteriormente se renombró como *World Wide Web* (red mundial), durante su implementación en 1990. Desarrollado sobre los protocolos existentes TCP e IP, está basado en cuatro bloques:

- Un formato de texto para representar documentos de hiper-texto: *HyperText Markup Language* (HTML).
- Un protocolo sencillo para el intercambio de esos documentos, del inglés: *HypertText Transfer Protocol* (HTTP) : protocolo de transferencia de hiper-texto.
- Un cliente que muestre (e incluso pueda editar) esos documentos. El primer navegador Web, llamado: *WorldWideWeb*.
- Un servidor para dar acceso a los documentos, una versión temprana: *httpd (http daemon)*

Estos cuatro bloques fundamentales se finalizaron para finales de 1990, y los primeros servidores estaban ya funcionando fuera del CERN a principios del 1991. El 6 de Agosto de 1991, el post de Tim Berners-Lee, se considera actualmente como el inicio oficial de la Web como proyecto público.

La versión del protocolo HTTP usada en aquel momento, era en realidad muy sencilla, posteriormente pasó a HTTP/0.9, referido algunas veces, como el protocolo de una sola línea.

HTTP/0.9 – El protocolo de una sola línea

La versión inicial de HTTP no tenía número de versión; aunque posteriormente se la denominó como 0.9 para distinguirla de las versiones siguientes. HTTP/0.9 es un protocolo extremadamente sencillo: una petición consiste simplemente en una única línea, que comienza por el único método posible `GET`, seguido por la dirección del recurso a pedir (no la URL, ya que tanto el protocolo, el servidor y el puerto, no son necesarios una vez ya se ha conectado al servidor).

```
GET /miPaginaWeb.html
```

La respuesta también es muy sencilla: solamente consiste el archivo pedido.

```
<HTML>

Una pagina web muy sencilla

</HTML>
```

Al contrario que sus posteriores evoluciones, el protocolo HTTP/0.9 no usa cabeceras HTTP, con lo cual únicamente es posible transmitir archivos HTML, y ningún otro tipo de archivos. Tampoco había información del estado ni códigos de error: en el caso un problema, el archivo HTML pedido, era devuelto con una descripción del problema dentro de él, para que una persona pudiera analizarlo.

HTTP/1.0 – Desarrollando expansibilidad

La versión HTTP/0.9 era ciertamente limitada y tanto los navegadores como los servidores, pronto ampliaron el protocolo para que fuera más flexible.

- La versión del protocolo se envía con cada petición: HTTP/1.0 se añade a la línea de la petición GET.
- Se envía también un código de estado al comienzo de la respuesta, permitiendo así que el navegador pueda responder al éxito o fracaso de la petición realizada, y actuar en consecuencia (como actualizar el archivo o usar la caché local de algún modo).
- El concepto de cabeceras de HTTP se presentó tanto para las peticiones como para las respuestas, permitiendo la trasmisión de meta-data y conformando un protocolo muy versátil y ampliable.
- Con el uso de las cabeceras de HTTP, se pudieron transmitir otros documentos además de HTML, mediante la cabecera `Content-Type`.

Una petición normal, sigue la estructura:

```
GET /mypage.html HTTP/1.0

User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)



200 OK

Date: Tue, 15 Nov 1994 08:12:31 GMT

Server: CERN/3.0 libwww/2.17

Content-Type: text/html
```

```
<HTML>

Una pagina web con una imagen

  <IMG SRC="/miImagen.gif">

</HTML>
```

Continua con una segunda conexión y la petición de una imagen:

```
GET /myImagen.gif HTTP/1.0

User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)



200 OK

Date: Tue, 15 Nov 1994 08:12:32 GMT

Server: CERN/3.0 libwww/2.17

Content-Type: text/gif

(image content)
```

Estas innovaciones, no se desarrollaron de forma planeada, sino más bien con una aproximación de prueba y error, entre los años 1991 y 1995: un servidor y un navegador, añadían una nueva funcionalidad y se evaluaba su aceptación. Debido a esto, en ese periodo eran muy comunes los problemas de interoperatividad. En Noviembre de 1996, para poner fin a estos problemas se publicó un documento informativo que describía las prácticas adecuadas, RFC 1945. Esté documento es la definición del protocolo HTTP/1.0. Resulta curioso, que realmente no es un estándar oficial.

HTTP/1.1 – El protocolo estándar.

En paralelo al uso, un poco desordenado, y las diversas implementaciones de HTTP/1.0, y desde el año 1995, un año antes de la publicación del documento del HTTP/1.0, un proceso de estandarización formal ya estaba en curso. La primera versión estandarizada de HTTP: el protocolo HTTP/1.1, se publicó en 1997, tan solo unos meses después del HTTP/1.0

HTTP/1.1 aclaró ambigüedades y añadió numerosas mejoras:

- Una conexión podía ser reutilizada, ahorrando así el tiempo de re-abrirla repetidas veces para mostrar los recursos empotrados dentro del documento original pedido.
- Enrutamiento ('Pipelining' en inglés) se añadió a la especificación, permitiendo realizar una segunda petición de datos, antes de que fuera respondida la primera, disminuyendo de este modo la latencia de la comunicación.
- Se permitió que las respuestas a peticiones podían ser divididas en sub-partes.
- Se añadieron controles adicionales a los mecanismos de gestión de la cache.
- La negociación de contenido, incluyendo el lenguaje, el tipo de codificación, o tipos, se añadieron a la especificación, permitiendo que servidor y cliente, acordasen el contenido más adecuado a intercambiarse.
- Gracias a la cabecera, `Host`, pudo ser posible alojar varios dominios en la misma dirección IP.

El flujo normal de una serie de peticiones y respuestas, bajo una única conexión, se expone a continuación:

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1

Host: developer.mozilla.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:50.0) Gecko/20100101 Firefox/50.0

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Referer: https://developer.mozilla.org/en-
US/docs/Glossary/Simple_header



200 OK

Connection: Keep-Alive

Content-Encoding: gzip
```

```
Content-Type: text/html; charset=utf-8

Date: Wed, 20 Jul 2016 10:55:30 GMT

Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"

Keep-Alive: timeout=5, max=1000

Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT

Server: Apache

Transfer-Encoding: chunked

Vary: Cookie, Accept-Encoding



(...contenido...)




GET /static/img/header-background.png HTTP/1.1

Host: developer.mozilla.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:50.0) Gecko/20100101 Firefox/50.0

Accept: */*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Referer: https://developer.mozilla.org/en-
US/docs/Glossary/Simple_header



200 OK

Age: 9578461
```

```
Cache-Control: public, max-age=315360000

Connection: keep-alive

Content-Length: 3077

Content-Type: image/png

Date: Thu, 31 Mar 2016 13:34:46 GMT

Last-Modified: Wed, 21 Oct 2015 18:27:50 GMT

Server: Apache



(image content of 3077 bytes)
```

HTTP/1.1 fue publicado inicialmente como RFC 2068 en Enero de 1997.

Más de 15 años de expansiones

Gracias a su expansibilidad - ya que la creación de nuevas cabeceras o métodos es sencilla - e incluso teniendo en cuenta que el protocolo HTTP/1.1 fue mejorado en dos revisiones: la primera, el documento RFC 2616, publicado en Junio de 1999 y posteriormente en los documentos RFC 7230-RFC 7235 publicados en Junio del 2014, en previsión de la publicación de HTTP/2. Así pues, el protocolo HTTP/1.1 ha sido increíblemente estable durante más de 15 años.

El uso de HTTP para transmisiones seguras

El mayor cambio en el desarrollo de HTTP fue a finales de 1994. En vez de trasmitir HTTP sobre la capa de TCP/IP, se creó una capa adicional sobre esta: SSL. La versión SSL 1.0 nunca fue publicada fuera de las compañías desarrolladoras, pero el SSL 2.0 y sus sucesoras SSL 3.0 y SSL 3.1 permitieron la creación del comercio electrónico en la Web (e-commerce), encriptando y garantizando la autenticidad de los mensajes intercambiados entre servidor y cliente. SSL se añadió a la lista de estándares y posteriormente evolucionó hasta ser el protocolo TLS, con versiones 1.0, 1.1 y 1.2, que fueron apareciendo para resolver vulnerabilidades. Actualmente se está desarrollando el protocolo TLS 1.3.

Durante el mismo periodo, la necesidad por una capa de trasporte encriptada aumentó; la Web, que permitía una relativa confianza de lo que era una mayoría

de trabajo académico, pasó a ser una jungla donde anuncios, individuos aleatorios o criminales competían para obtener tanta información privada sobre la gente como pudieran, o trataban de suplantarlos o incluso sustituir los datos trasmitidos por otros alterados. A medida que hubo aplicaciones que se desarrollaban y funcionaban sobre HTTP, fueron más y más funcionales, tener acceso a más y más información personal como contactos, e-mails, o posición geográfica del usuario, la necesidad de tener el protocolo TLS, fue fundamental incluso fuera del ámbito del comercio electrónico.

Uso de HTTP para aplicaciones complejas

La visión original de Tim Berners-Lee para la Web no era solo un medio de 'solo' lectura. Él había visionado una Web donde la gente pudiese añadir y mover documentos de forma remota, un estilo de sistema de archivos distribuido. Sobre el año 1996, HTTP se había desarrollado para permitir la autoría, y fue creado un estándar denominado WebDAB. Este fue más tarde ampliado por aplicaciones específicas como CardDAV, para permitir libros de direcciones, y CalDAV para trabajar con calendarios. Pero todos estas extensiones '*DAV', tenían una debilidad, y es que debían ser implementadas por los servidores, para poder ser usadas, lo cual era bastante complejo. Así pues, su uso en la Web fue bastante acotado.

En el año 2000, un nuevo formato para usar HTTP fue diseñado: REST (del inglés: 'Representational State Transfer'). Las acciones de la nueva API no estaban supeditadas a nuevos métodos HTTP, unicamente al acceso a URIs especificas con métodos HTTP/1.1). Esto permitió que cualquier aplicación Web dispusiera de una API, para permitir la recuperación y modificación de datos, sin tener que actualizar servidores o navegadores; todo lo que se necesitaba era incluido en los archivos servidos por los sitios Web. La contrapartida del modelo REST está en que cada sitio Web define su propia versión no estándar de API RESTful y tiene un control total sobre ella; al contrario del formato *DAV donde clientes y servidores eran interoperables. La arquitectura REST empezó a ser muy común a partir del año 2010.

Desde el año 2005, las APIs disponibles para páginas Web a aumentado considerablemente, y muchas de estas nuevas APIs dependen de cabeceras HTTP específicas para funciones concretas:

- Eventos enviados por el servidor: El servidor es el que ocasionalmente inicia los mensajes hacia el navegador.
- WebSocket, un nuevo protocolo que puede establecerse actualizando una conexión HTTP existente.

Relajación del modelo de seguridad de la Web

El protocolo HTTP es independiente del modelo de seguridad de la Web: la política del mismo origen. De hecho, el actual modelo de seguridad de la Web, ha sido desarrollado con posterioridad a la creación del protocolo HTTP.

A lo largo de los años, se ha probado útil, poder ser más permisivo con ella, permitiendo que bajo ciertos requerimientos se puedan levantar algunas de las restricciones de esta política. Cuanto y cuantas de estas restricciones se pueden saltar es comunicado desde el servidor al cliente, mediante una serie de nuevas cabeceras HTTP. Estas están especificadas en los documentos como CORS (del inglés Cross-Origin Resource Sharing, que viene a significar: recursos compartidos de orígenes cruzados) y el CSP (del inglés: Content Security Policy, que traducido es: política de seguridad de contenidos).

Además de estas ampliaciones, muchas otras cabeceras han sido añadidas, algunas únicamente experimentales. Algunas de ellas notables son: Do Not Track (DNT (en-US)); cabecera de control de privacidad: `X-Frame-Options`, y Upgrade-Insecure-Requests (en-US).

HTTP/2 – Un protocolo para un mayor rendimiento

A lo largo de los años, las páginas Web han llegado a ser mucho más complejas, incluso llegando a poder considerarse como aplicaciones por derecho propio. La cantidad de contenido visual, el tamaño de los scripts, y los scripts que añaden interactividad ha aumentado mucho también. Muchísimos más datos son transmitidos bajo muchas más peticiones HTTP. Las conexiones HTTP/1.1 han de enviar las peticiones HTTP en el orden correcto. Teóricamente, sería posible usar varias conexiones en paralelo (normalmente entre 5 y 8), aumentando consecuentemente la complejidad del proceso. Por ejemplo, el HTTP 'pipelining' ha demostrado ser un lastre para el desarrollo Web.

En la primera mitad de la década de 2010, Google demostró un proceso alternativo para el intercambio de data entre clientes y servidores, implementando el protocolo experimental SPDY (pronunciado como en inglés *'speedy'*). Este atrajo mucho interés por los desarrolladores de tanto los navegadores como los servidores. Definiendo una mejora en los tiempos de respuesta, y resolviendo el problema de datos duplicados transmitidos. SPDY sirvió como base para el desarrollo del protocolo HTTP/2.

El protocolo HTTP/2, tiene notables diferencias fundamentales respecto a la versión anterior HTTP/1.1

- Es un protocolo binario, en contraposición a estar formado por cadenas de texto, tal y como estaban basados sus protocolos anteriores. Así pues, no se puede leer directamente, ni crear manualmente A pesar de este inconveniente, gracias a este cambio es posible utilizar en él técnicas de optimización.
- Es un protocolo multiplexado. Peticiones paralelas pueden hacerse sobre la misma conexión, no está sujeto pues a mantener el orden de los mensajes, ni otras restricciones que tenían los protocolos anteriores HTTP/1.x

- Comprime las cabeceras, ya que estas, normalmente son similares en un grupo de peticiones. Esto elimina la duplicación y retardo en los datos a transmitir.
- Esto permite al servidor almacenar datos en la caché del cliente, previamente a que estos sean pedidos, mediante un mecanismo denominado '*server push*'.

Estandarizado de manera oficial en Mayo de 2015, HTTP/2 ha conseguido muchos éxitos. En Julio de 2016, un 8.7% de todos los sitios Web[1] estaban usándolo ya, representando más del 68% de todo su tráfico[2]. Los sitios Web con mucho tráfico, fueron aquellos que lo adoptaron más rápidamente, ahorrando considerablemente las sobrecargas en la transferencia de datos, ... y en sus presupuestos.

Esta rápida adopción era esperada, ya que el uso de HTTP/2, no requiere de una adaptación de los sitios Web y aplicaciones: el uso de HTTP/1.1 o HTTP/2 es transparente para ellos. El uso de un servidor actual, comunicándose con un navegador actualizado, es suficiente para permitir su uso: únicamente en casos particulares fue necesario impulsar su utilización; y según se actualizan servidores y navegadores antiguos, su utilización aumenta, sin que requiera un mayor esfuerzo de los desarrolladores Web.

Post-evolución del HTTP/2

Con la publicación de la versión del protocolo HTTP/2, esté no ha dejado de evolucionar. Como con el HTTP/1.x, anteriormente, la extensibilidad del HTTP se sigue usando para añadir nuevas funcionalidades. Podemos enumerar algunas de estas nuevas características que se desarrollaron en el año 2016:

- Soporte de la cabecera Alt-Svc (en-US), la cual permite disociar la identificación de una ubicación, con respecto a un recurso pedido, permitiendo el uso más inteligente de los mecanismos de cacheo de memoria de los CDN.
- La introducción de la cabecera `Client-Hints`, que permite al navegador, o cliente, comunicar proactivamente al servidor, sus necesidades o restricciones de hardware.
- La introducción de prefijos de seguridad en la cabecera `Cookie`, esto ayuda a garantizar que una cookie, no ha sido alterada.

Esta evolución del HTTP demuestra su capacidad de ampliación y simplicidad, permitiendo así de forma deliberada su uso para muchas aplicaciones y favoreciendo el uso de este protocolo. El entorno en el que el HTTP se usa hoy en día es muy distinto al que había a principios de la década de 1990. El desarrollo original de HTTP ha demostrado ser una obra maestra, permitiendo a la Web evolucionar a lo largo de un cuarto de siglo, sin la necesidad de un 'amotinamiento'. Corrigiendo errores, y manteniendo la flexibilidad y

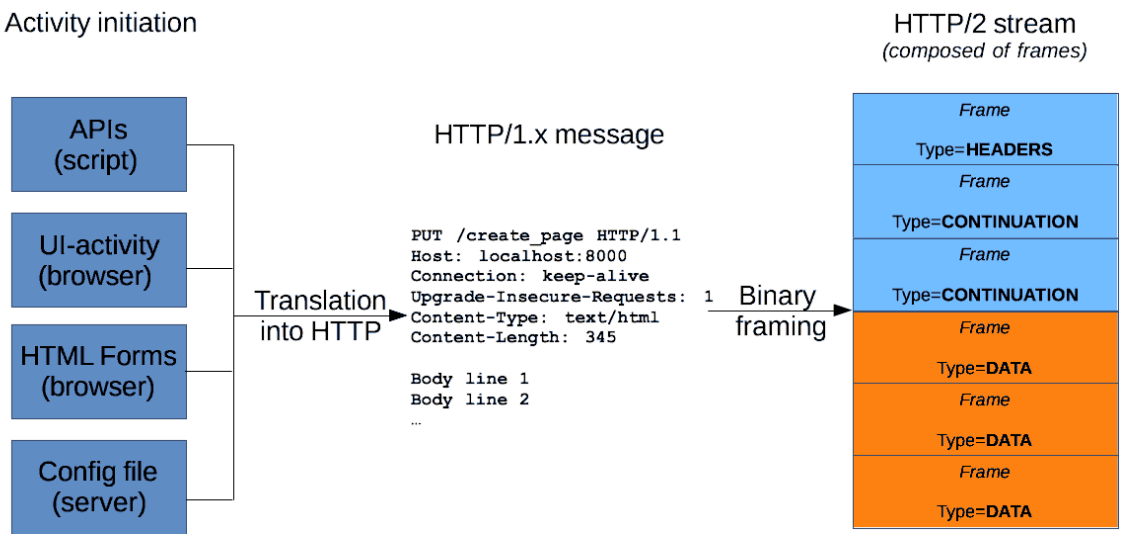
extensibilidad que han hecho al HTTP un éxito, la adopción del HTTP/2 tiene un brillante futuro.

Mensajes HTTP

Los mensajes HTTP, son los medios por los cuales se intercambian datos entre servidores y clientes. Hay dos tipos de mensajes: *peticiones*, enviadas por el cliente al servidor, para pedir el inicio de una acción; y *respuestas*, que son la respuesta del servidor.

Los mensajes HTTP están compuestos de texto, codificado en ASCII, y pueden comprender múltiples líneas. En HTTP/1.1, y versiones previas del protocolo, estos mensajes eran enviados de forma abierta a través de la conexión. En HTTP/2.0 los mensajes, que anteriormente eran legibles directamente, se conforman mediante tramas binarias codificadas para aumentar la optimización y rendimiento de la transmisión.

Los desarrolladores de páginas Web, o administradores de sitios Web, desarrolladores... raramente codifican directamente estos mensajes HTTP. Normalmente especifican estos mensajes HTTP, mediante archivos de configuración (para proxies, y servidores), APIs (para navegadores) y otros medios.

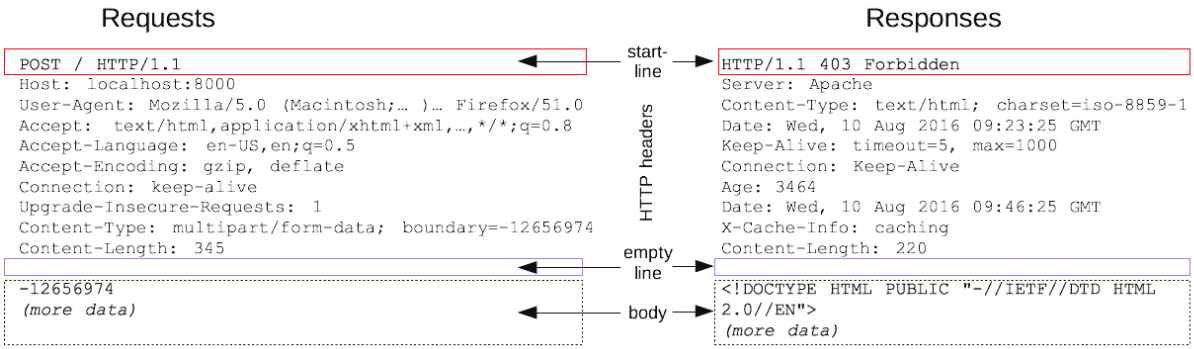


El mecanismo de tramas binarias de HTTP/2 ha sido diseñado para que no necesite ninguna modificación de las APIs o archivos de configuración utilizados: es totalmente transparente para el usuario.

Las peticiones y respuestas HTTP, comparten una estructura similar, compuesta de:

1. Una *línea de inicio* ('*start-line*' en inglés) describiendo la petición a ser implementada, o su estado, sea de éxito o fracaso. Esta línea de comienzo es siempre una única línea.
2. Un grupo opcional de *cabeceras HTTP*, indicando la petición o describiendo el cuerpo ('*body*' en inglés) que se incluye en el mensaje.
3. Una línea vacía ('*empty-line*' en inglés) indicando toda la metainformación ha sido enviada.
4. Un campo de cuerpo de mensaje opcional ('*body*' en inglés) que lleva los datos asociados con la petición (como contenido de un formulario HTML), o los archivos o documentos asociados a una respuesta (como una página HTML, o un archivo de audio, vídeo ...) . La presencia del cuerpo y su tamaño es indicada en la línea de inicio y las cabeceras HTTP.

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la *cabeza* de las peticiones, mientras que su contenido en datos se conoce como el *cuerpo* del mensaje.



Peticiones HTTP

Línea de inicio

Las peticiones HTTP son mensajes enviados por un cliente, para iniciar una acción en el servidor. Su línea de inicio está formada por tres elementos:

1. Un *método HTTP*, un verbo como: `GET`, `PUT` o `POST`) o un nombre como: HEAD (en-US) o OPTIONS (en-US)), que describan la acción que se pide sea realizada. Por ejemplo, `GET` indica que un archivo ha de ser enviado hacia el cliente, o `POST` indica que hay datos que van a ser enviados hacia el servidor (creando o modificando un recurso, o generando un documento temporal para ser enviado).

2. El objetivo de una petición normalmente es una URL, o la dirección completa del protocolo, puerto y dominio también suelen ser especificados por el contexto de la petición. El formato del objetivo de la petición varía según los distintos métodos HTTP. Puede ser:

    - Una dirección absoluta, seguida de un signo de cierre de interrogación '?' y un texto de consulta. Este es el formato más común, conocido como el formato original ('*origin form*' en inglés), se usa en los métodos `GET`, `POST`, `HEAD`, y `OPTIONS` .
      ```
      POST / HTTP 1.1
      GET /background.png HTTP/1.0
      HEAD /test.html?query=alibaba HTTP/1.1
      OPTIONS /anypage.html HTTP/1.0
      ```
    - Una URL completa; conocido como el formato absoluto, usado mayormente con `GET` cuando se conecta a un proxy.
      ```
      GET http://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1
      ```
    - El componente de autoridad de una URL, formado por el nombre del dominio y opcionalmente el puerto (el puerto precedido por el símbolo `':'`), se denomina a este formato como el formato de autoridad. Únicamente se usa con CONNECT cuando se establece un túnel HTTP.
      ```
      CONNECT developer.mozilla.org:80 HTTP/1.1
      ```
    - El formato de asterisco, se utiliza un asterisco (`'*'`) junto con las opciones: `OPTIONS` , representando al servidor entero en conjunto.
      ```
      OPTIONS * HTTP/1.1
      ```

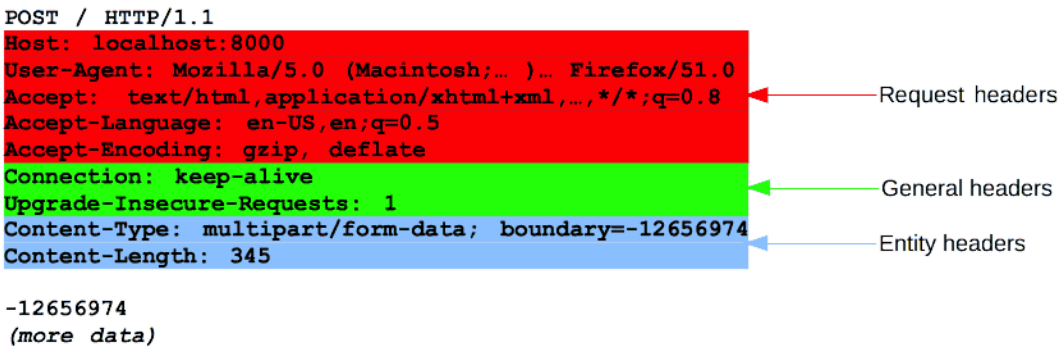
3. la versión de HTTP, la cual define la estructura de los mensajes, actuando como indicador, de la versión que espera que se use para la respuesta.

Cabeceras

Las cabeceras HTTP de una petición siguen la misma estructura que la de una cabecera HTTP. Una cadena de caracteres, que no diferencia mayúsculas ni minúsculas, seguida por dos puntos (`':'`) y un valor cuya estructura depende de la cabecera. La cabecera completa, incluido el valor, ha de ser formada en una única línea, y pude ser bastante larga.

Hay bastantes cabeceras posibles. Estas se pueden clasificar en varios grupos:

- *Cabeceras generales,* ('*General headers*' en inglés), como Via (en-US), afectan al mensaje como una unidad completa.
- Cabeceras de petición, ('*Request headers*' en inglés), como `User-Agent`, `Accept-Type`, modifican la petición especificándola en mayor detalle (como: Accept-Language (en-US), o dándole un contexto, como: `Referer`, o restringiéndola condicionalmente, como: `If-None`.
- *Cabeceras de entidad, ('Entity headers'* en inglés), como `Content-Length` las cuales se aplican al cuerpo de la petición. Por supuesto, esta cabecera no necesita ser transmitida si el mensaje no tiene cuerpo ('*body*' en inglés).



Cuerpo

La parte final de la petición en el cuerpo. No todas las peticiones llevan uno: las peticiones que reclaman datos, como `GET`, `HEAD`, `DELETE`, o `OPTIONS`, normalmente, no necesitan ningún cuerpo. Algunas peticiones pueden mandar peticiones al servidor con el fin de actualizarlo: como es el caso con la petición `POST` (que contiene datos de un formulario HTML).

Los cuerpos pueden ser divididos en dos categorías:

- Cuerpos con un único dato, que consisten en un único archivo definido por las dos cabeceras: `Content-Type` y `Content-Length`.
- Cuerpos con múltiples datos, que están formados por distintos contenidos, normalmente están asociados con los formularios HTML.

Respuestas HTTP

Línea de estado

La línea de inicio de una respuesta HTTP, se llama la *línea de estado*, y contienen la siguiente información:

1. La *versión del protocolo*, normalmente `HTTP/1.1`.

2. Un *código de estado*, indicando el éxito o fracaso de la petición. Códigos de estado muy comunes son: `200`, `404`, o `302`
3. Un *texto de estado*, que es una breve descripción, en texto, a modo informativo, de lo que significa el código de estado, con el fin de que una persona pueda interpretar el mensaje HTTP.

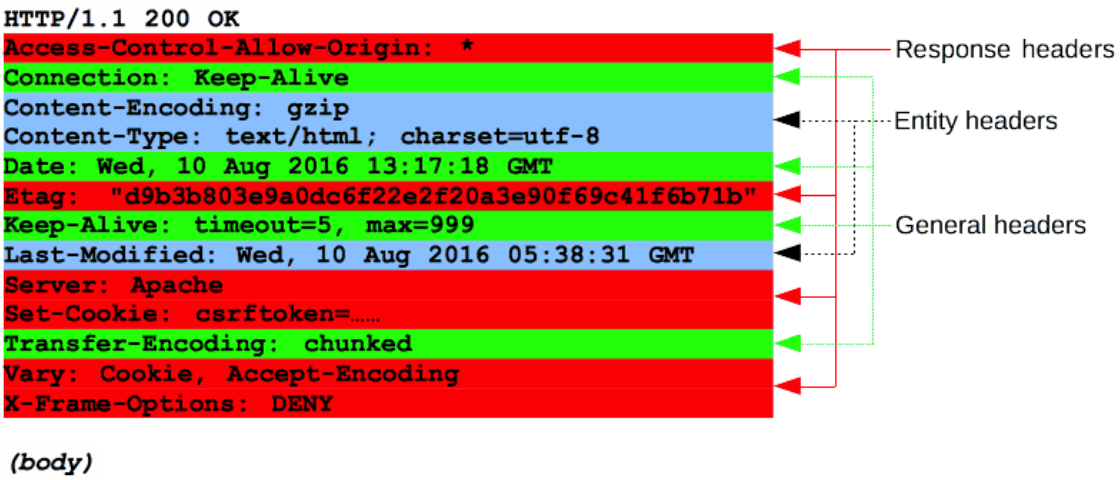
Una línea de estado típica es, por ejemplo: `HTTP/1.1 404 Not Found.`

Cabeceras

Las cabeceras HTTP para respuestas siguen también la misma estructura como cualquier otra cabecera: una cadena de texto, que no diferencia entre mayusculas y minúsculas, seguida por dos puntos (`':'`) y un valor cuya estructura depende del tipo de cabecera. Toda la cabecera incluido su valor, se ha de expresar en una única línea.

Existen varias cabeceras posibles. Estas se pueden dividir en distintos grupos:

- *Cabeceras generales,* ('*General headers*' en inglés), como Via (en-US), afectan al mensaje completo.
- Cabeceras de petición, ('*Request headers*' en inglés), como `Vary` , `Accept-Ranges`, dan información adicional sobre el servidor, que no tiene espacio en la línea de estado.
- *Cabeceras de entidad, ('Entity headers'* en inglés), como `Content-Length` las cuales se aplican al cuerpo de la petición. Por supuesto, esta cabecera no necesita ser transmitida si el mensaje no tiene cuerpo ('*body*' en inglés).



Cuerpo

La última parte del mensaje de respuesta él es 'cuerpo'. No todas las respuestas tienen uno, respuestas con un código de estado como `201` o `204` normalmente prescinden de él.

De forma general, los cuerpos se pueden diferenciar en tres categorías:

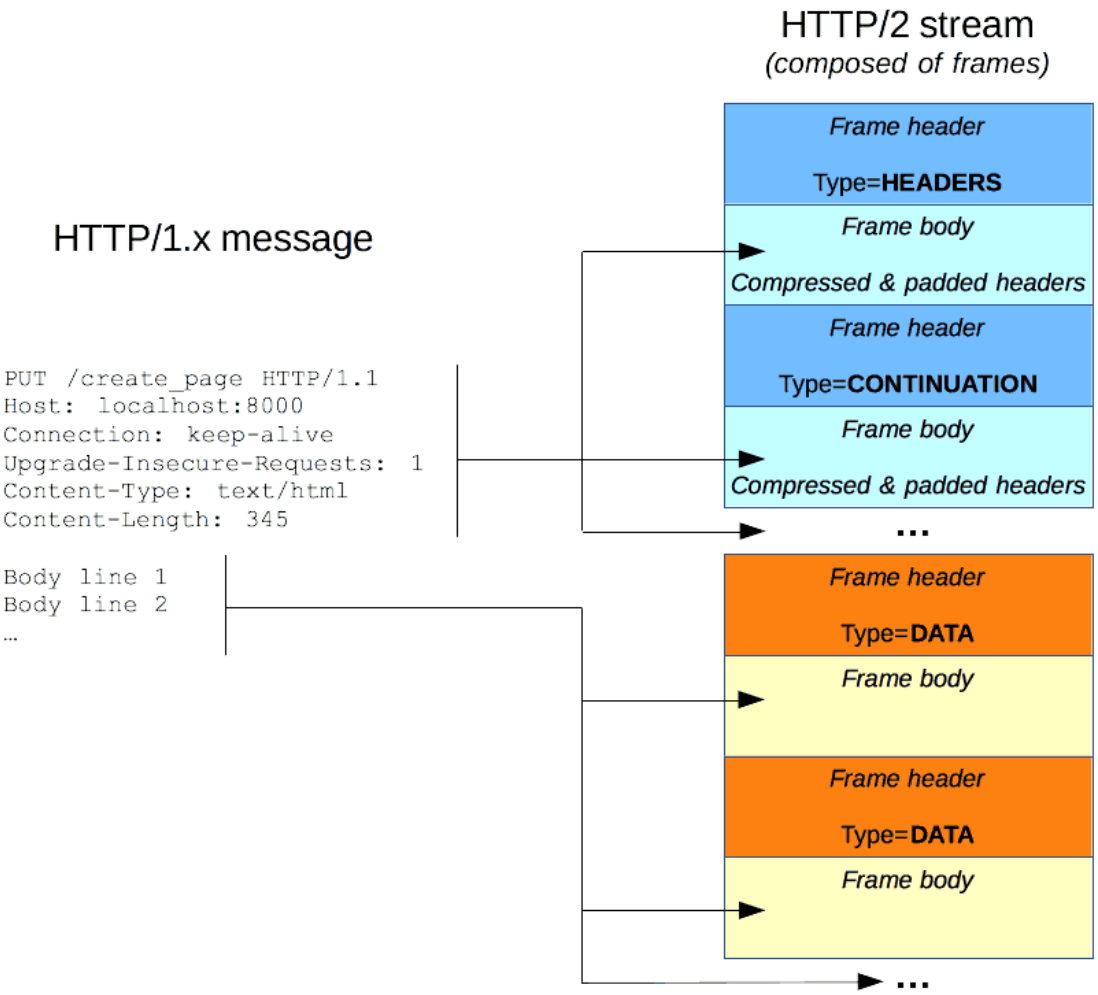
- Cuerpos con un único dato, consisten en un simple archivo, de longitud conocida y definido en las cabeceras: `Content-Type` y `Content-Length`.
- Cuerpos con un único dato, consisten en un simple archivo, de longitud desconocida, y codificado en partes, indicadas con `Transfer-Encoding` valor `chunked` (que significa: 'partido' en inglés).
- Cuerpos con múltiples datos, consisten en varios datos, cada uno con una sección distinta de información. Este caso es relativamente raro y poco común.

Tramas HTTP/2

Los mensajes HTTP/1.x tienen algunas desventajas por su no muy alta eficiencia en la transmisión.

- Las cabeceras, al contrario de los cuerpos, no se comprimen.
- Las cabeceras, habitualmente se repiten de un mensaje al siguiente, aun así, la cabecera se repite en todos los mensajes.
- No se puede multiplexar. Se han de abrir varias conexiones para el mismo servidor, las conexiones TCP 'en caliente' ('*warm TCP connections*' en inglés) son más eficientes que las conexiones 'en frio'.

HTTP/2 introduce un paso extra: divide los mensajes HTTP/1.x en tramas que integra en un flujo de datos. Los datos y las tramas de las cabeceras se separan, esto permite la compresión de las cabeceras. Varios flujos de datos pueden combinarse juntos, y entonces se puede usar un procedimiento de multiplexación, permitiendo un uso más eficiente, de las conexiones TCP.



Las tramas HTTP son transparentes para los desarrolladores Web. Este paso adicional en HTTP/2, de los mensajes HTTP/1.0 y el protocolo por debajo. No son necesarios cambios en las APIs usadas por los desarrolladores Web para utilizar estas tramas HTTP, cuando las usan ambos: servidor y navegador.

Conclusión

Los mensajes HTTP son la clave para usar HTTP; su estructura es sencilla y son fácilmente ampliables. El protocolo HTTP/2 añade un mecanismo de tramas y una capa intermedia entre la sintaxis de HTTP/1.x y su protocolo inferior, sin modificarlo radicalmente: se construye sobre mecanismos de transmisión probados.

Una típica sesión de HTTP

En los protocolos basados en el modelo cliente-servidor, como es el caso del HTTP, una sesión consta de tres fases:

1. El cliente establece una conexión TCP (o la conexión correspondiente si la capa de transporte corresponde a otro protocolo).
2. El cliente manda su petición, y espera por la respuesta.
3. El servidor procesa la petición, y responde con un código de estado y los datos correspondientes.

A partir del protocolo HTTP/1.1 la conexión, no se cierra al finalizar la tercera fase, y el cliente puede continuar realizando peticiones. Esto significa que la segunda y tercera fase, pueden repetirse cualquier número de veces.

Estableciendo una conexión

En un protocolo cliente servidor, es siempre el cliente el que establece la conexión. Iniciar una conexión en HTTP, implica iniciar una conexión en el protocolo correspondiente a la capa de comunicación subyacente, que normalmente es TCP.

En TCP el puerto por defecto, para un servidor HTTP en un computador, es el puerto 80. Se pueden usar otros puertos como el 8000 o el 8080. La URL de la página pedida contiene tanto el nombre del dominio, como el número de puerto, aunque este puede ser omitido, si se trata del puerto 80.

Nota: El modelo cliente-servidor no permite que el servidor mande datos al cliente sin una petición explicita. Como solución parcial a este problema, los desarrolladores web, usan varias técnicas, como hacer un ping al servidor periódicamente, mediante `XMLHTTPRequest`, `Fetch` APIs, o usar la HTML WebSockets API o protocolos similares.

Mandando una petición

Una vez la conexión está establecida, el cliente, puede mandar una petición de datos (normalmente es un navegador, u otra cosa, como una 'araña' (o 'crawler' en inglés), un sistema de indexación automático de páginas web). La petición de datos de un cliente HTTP, consiste en directivas de texto, separadas mediante CRLF (retorno de carro, y cambio de linea), y se divide en tres partes:

1. La primera parte, consiste en una línea, que contiene un método, seguido de sus parámetros:

    - la dirección del documento pedido: por ejemplo, su URL completa, sin indicar el protocolo o el nombre del dominio.
    - la versión del protocolo HTTP

2. La siguiente parte, está formada por un bloque de líneas consecutivas, que representan las cabeceras de la petición HTTP, y dan información al servidor, sobre qué tipo de datos es apropiado (como qué lenguaje usar, o el tipo MIME a usar), u otros datos que modifiquen su comportamiento (como que no envié la respuesta si ya está cacheada). Estas cabeceras HTTP forman un bloque que acaba con una línea en blanco.

3. La parte final es un bloque de datos opcional, que puede contener más datos para ser usados por el método POST.

Ejemplo de peticiones

Si queremos una página web, como, por ejemplo: http://developer.mozilla.org/, y además le indicamos al servidor que se preferiría la página en Francés, si fuera posible:

```
GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr


```

Observe la línea vacía al final, que separa el bloque de datos, del bloque de cabecera. Como no existe el campo `Content-Length` en la cabecera de HTTP, el bloque de datos está vacío, y ahí está el fin de la cabecera, permitiendo al servidor procesar la petición en el momento que recibe la línea vacía.

Otro ejemplo, en el caso del envío de los datos de un formulario, la trama es:

```
POST /contact_form.php HTTP/1.1

Host: developer.mozilla.org

Content-Length: 64

Content-Type: application/x-www-form-urlencoded



```

```
name=Juan%20Garcia&request=Envieme%20uno%20de%20sus%20catalogos
```

Métodos de peticiones

HTTP define un conjunto de métodos de peticiones en los que se indican las acciones que se piden realizar al recibir un conjunto de datos. A pesar de que pueden referirse como 'nombres', estos métodos de petición son denominados a veces como 'verbos' de HTTP. Las peticiones más comunes son `GET` y `POST`:

- El método `GET` hace una petición de un recurso específico. Las peticiones con `GET` unicamente hacen peticiones de datos.
- El método `POST` envía datos al servidor de manera que este pueda cambiar su estado. Este es el método usado normalmente para enviar los datos de un formulario HTML.

Estructura de la respuesta del servidor

Después de que el agente de usuario envía su petición, el servidor web lo procesa, y a continuación responde. De forma similar a la petición del servidor, la respuesta del servidor está formada por directivas de texto, separadas por el carácter CRLF, y divida en tres bloques.

1. La primera línea, es la línea de estado, que consiste en una confirmación de la versión de HTTP utilizado, y seguido por el estado de la petición (y una breve descripción de este, en formato de texto, que pueda ser leído por personas).
2. Las líneas siguientes representan cabeceras de HTTP concretas, dando al cliente información sobre los datos enviado(por ejemplo, sy tipo, su tamaño, algoritmos de compresión utilizados, y sugerencias para el cacheo). Al igual que las cabeceras HTTP de la petición de un cliente, las cabeceras HTTP del servidor, finalizan con una línea vacía.
3. El bloque final, es el bloque que puede contener opcionalmente los datos.

Ejemplos de respuestas

La respuesta correcta de una página web, es como sigue:

```
HTTP/1.1 200 OK

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

ETag: "51142bc1-7449-479b075b2891b"
```

```
Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html



<!DOCTYPE html... (aquí estarían los 29769 bytes de la página web pedida)
```

La respuesta para la petición de datos que han sido movidos permanentemente sería:

```
HTTP/1.1 301 Moved Permanently

Server: Apache/2.2.3 (Red Hat)

Content-Type: text/html; charset=iso-8859-1

Date: Sat, 09 Oct 2010 14:30:24 GMT

Location: https://developer.mozilla.org/ (este es el nuevo enlace a los datos; se espera que el agente de usuario lo pida a continuación)

Keep-Alive: timeout=15, max=98

Accept-Ranges: bytes

Via: Moz-Cache-zlb05

Connection: Keep-Alive

X-Cache-Info: caching

X-Cache-Info: caching

Content-Length: 325 (se da una página por defecto para mostrar en el caso de que el agente de usuario no sea capaz de seguir el enlace)



<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
```

```
<html><head>

<title>301 Movido permanentemente</title>

</head><body>

<h1>Movido de forma permanente</h1>

<p>El documento ha sido movido <a
href="https://developer.mozilla.org/">aquí</a>.</p>

<hr>

<address>Apache/2.2.3 (Red Hat) Servidor en:
developer.mozilla.org Port 80</address>

</body></html>
```

Una notificación de que los datos pedidos no existen:

```
HTTP/1.1 404 Not Found

Date: Sat, 09 Oct 2010 14:33:02 GMT

Server: Apache

Last-Modified: Tue, 01 May 2007 14:24:39 GMT

ETag: "499fd34e-29ec-42f695ca96761;48fe7523cfcc1"

Accept-Ranges: bytes

Content-Length: 10732

Content-Type: text/html



<!DOCTYPE html... (contiene una página personalizada de ayuda al
usuario para que pueda encontrar los datos que busca)
```

Códigos de estado de las respuestas

Los códigos de estado de las respuestas indican si una petición HTTP ha sido finalizada correctamente. Las respuestas se agrupan en cinco clases: respuestas informativas, respuestas de finalización correcta, redirecciones, errores del cliente y errores del servidor.

- `200`: OK. La petición ha sido procesada correctamente
- `301`: Datos movidos permanentemente. Este código de respuesta indica que la URI de los datos pedidos ha cambiado.
- `404`: Datos no encontrados. El servidor no puede localizar los datos pedidos.
- `501`: Not Implemented. El servidor no soporta la funcionalidad.

Identificación de recursos web

El objetivo de una solicitud HTTP se denomina "recurso", (es decir: datos), y dicho recurso, no posee un tipo definido por defecto; puede ser un documento, o una foto, o cualquier otra posibilidad. Cada recurso es identificado por un Identificador Uniforme de Recursos (URI) y es utilizado a través de HTTP, para la identificación del tipo de recurso.

La identidad y la localización del recurso en la Web son en su mayoría proporcionados por una sola dirección URL (Localizador de Recursos Uniforme; un tipo de URI). A veces, el mismo URI no proporciona la identidad ni la ubicación: HTTP usa un encabezado HTTP especifico, Alt-Svc cuando el recurso solicitado por el cliente quiere acceder a él en otra ubicación.

URLs and URNs

URLs

La forma más común de URI es la (URL) (de las siglas en inglés: "*Uniform Resource Locator*", que podría traducirse como: Localizador Uniforme de Recursos), *que se conoce como la dirección web.*

```
https://developer.mozilla.org

https://developer.mozilla.org/en-US/docs/Learn/

https://developer.mozilla.org/en-US/search?q=URL
```

Cualquiera de estas URLs se puede escribir en la barra de direcciones de su navegador para decirle que cargue la página asociada (recurso).

Una URL está compuesta de diferentes partes, algunas obligatorias y otras son opcionales. Un ejemplo más complejo podría tener este aspecto:

```
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
```

URNs

Un URN es una URI que identifica un recurso por su nombre en un espacio de nombres particular.

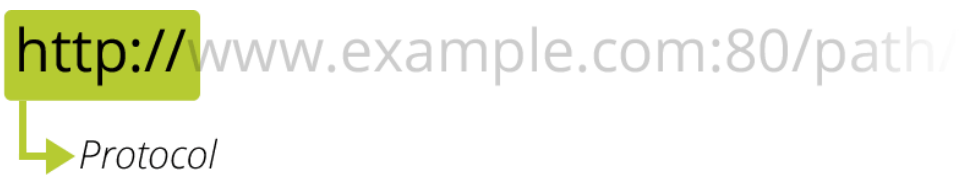
```
urn:isbn:9780141036144

urn:ietf:rfc:7230
```

Las dos URNs corresponden a

- El libro "1984" por George Orwell,
- La especificación IETF 7230, Hypertext Transfer Protocol (HTTP/1.1): Sintaxis de Mensajes y Enrutamiento.

Sintaxis de Identificador Uniforme de Recursos (URIs)

Esquema o protocolo



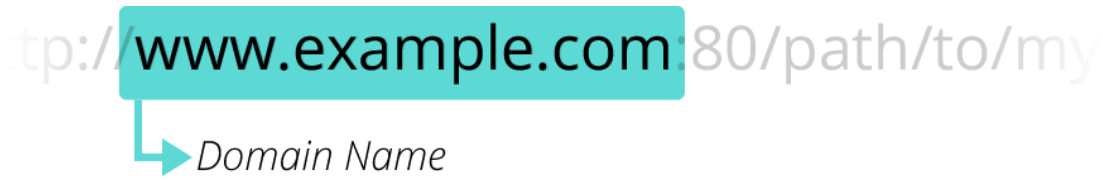
`http://` es el protocolo. Indica que el protocolo debe utilizar el navegador. Por lo general, es el protocolo HTTP o su versión segura, HTTPS. La Web requiere de uno de estos dos, pero los navegadores también saben cómo manejar otros protocolos como `mailto:` (para abrir un cliente de correo) o `ftp:` para manejar la transferencia de

archivos, por lo que no se sorprenda si usted ve este tipo de protocolos. Los esquemas comunes son:

Esquema	Descripción
data	Datos URIs
file	Host-nombre de archivo específicos
ftp	Protocolo de Transferencia de Archivos
http/https	Protocolo de transferencia de Hipertexto (Seguro)
mailto	Dirección de correo electrónico
ssh	shell seguro
tel	teléfono
urn	Nombres Uniformes de Recursos

Esquema	Descripción
view-source	Código fuente del recurso
ws/wss	(Encriptado) conexiones WebSocket

Autoridad



`www.example.com` es el nombre de dominio o autoridad que gobierna el espacio de nombres. Indica cuando es solicitado el servidor Web . Alternativamente, Es posible usar directamente una IP address, pero debido a que es menos conveniente, no se usa muy a menudo en la Web.

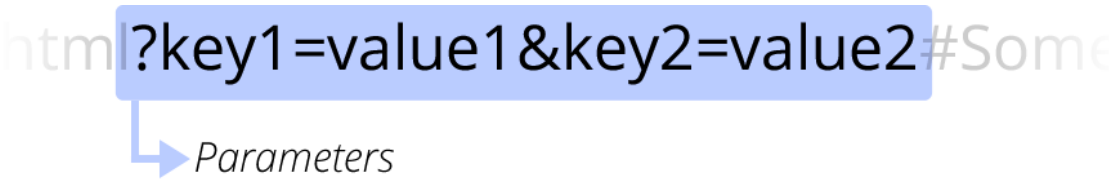
Puerto

`:80` es el puerto en este caso. Indica la técnica "puerta" usada para acceder a los recursos en el servidor web. Usualmente es omitido si el servidor web usa los puertos estándares del protocolo HTTP (80 para HTTP y 443 para HTTPS) para permitir el acceso a sus recursos. De lo contrario, es obligatorio.

Ruta de Acceso

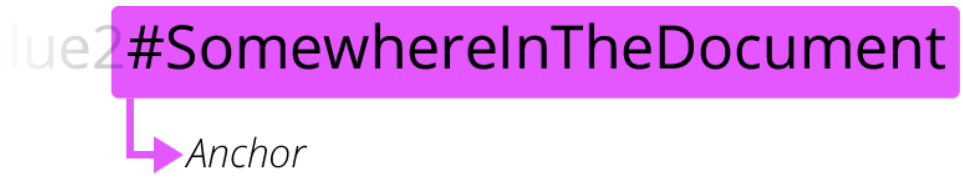
`/path/to/myfile.html` es la ruta de acceso al recurso en el servidor Web. En los primeros días de la Web, una ruta como esta presentaba la ubicación física del archivo en el servidor Web. Hoy en día, es sobre todo una abstracción manejada por los servidores Web sin ningún tipo de realidad física.

Consulta



`?key1=value1&key2=value2` son unos parámetros adicionales proporcionados al servidor Web. Esos parámetros son una lista de pares llave/valores separados por el símbolo &. El servidor Web puede utilizar estos parámetros para hacer cosas adicionales antes de retornar el recurso al usuario. Cada servidor Web tiene sus propias reglas con respecto a los parámetros, y la única manera confiable de saber cómo un servidor web especifico está manejando parámetros es preguntando al usuario del servidor web.

Fragmento



`#SomewhereInTheDocument` es una referencia a otra parte del propio recurso. Esto representa una especie de "marcador" dentro del

recurso, otorgándole al navegador las instrucciones para mostrar el contenido que se encuentra en esa referencia señalada. En un documento HTML, por ejemplo, el navegador se desplazará hasta el punto donde se define el fragmento; en un video o documento de audio, el navegador intentará ir a la vez que el ancla se presenta. Vale la pena señalar que la parte después de la #, también conocido como identificador de fragmento, nunca se envía al servidor con la solicitud.

Ejemplos

```
https://developer.mozilla.org/en-US/docs/Learn

tel:+1-816-555-1212

git@github.com:mdn/browser-compat-data.git

ftp://example.org/resource.txt

urn:isbn:9780141036144
```

Especificaciones

Especificación	Título
RFC 7230, section 2.7: Uniform Resource Identifiers	Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

Datos URIs

Datos URIs, URLs prefijados con los datos`:` esquema, permiten a los creadores de contenido incorporar pequeños archivos en línea en los documentos.

Sintaxis

Los datos URIs se componen de cuatro partes: un prefijo (`data:`), un tipo MIME que indica el tipo de datos, un token `base64` opcional no textual, y los datos en sí:

```
data:[<mediatype>][;base64],<data>
```

El `mediatype` es una cadena de tipo MIME, por ejemplo `'image/jpeg'` para un archivo de imagen JPEG. Si se omite, será por defecto `text/plain;charset=US-ASCII`

Si el dato es textual, solo tiene que insertar el texto (utilizando las entidades o escapes adecuados en función del tipo de documento). Por otra parte, puedes especificar base-64 para insertar datos binarios codificados en base-64.

Algunos ejemplos:

`data:,Hello%2C%20World!`
: Datos simples text/plain

`data:text/plain;base64,SGVsbG8sIFdvcmxkIQ%3D%3D`
: versión codificada en base64-encoded de las anteriores

`data:text/html,%3Ch1%3EHello%2C%20World!%3C%2Fh1%3E`
: Un documento HTML con `<h1>Hello, World!</h1>`

`data:text/html,<script>alert('hi');</script>`
: Un documento HTML que ejecuta una alerta Javascript. Tenga en cuenta que se requiere la etiqueta script de cierre.

Codificación de datos en formato base64

Esto se puede hacer fácilmente desde la línea de comandos usando `uuencode,` una utilidad disponible en sistemas Linux y Mac OS X:

```
uuencode -m infile remotename
```

El parámetro `infile` es el nombre para el archivo que desees decodificar en formato base64, y `remotename` es el nombre remoto para el archivo, que no se utilizará realmente en los datos de las URLs.

La salida será similar a esto:

```
xbegin-base64 664 test

YSBzbGlnaHRseSBsb25nZXIgdGVzdCBmb3IgdGV2ZXIK

====
```

El URI de datos utilizará los datos codificados después de la cabecera inicial.

En la página Web, usando JavaScript

Las Web tiene APIs primitivas para codificar o decodificar en base64: codificación y decodificación Base64.

Problemas comunes

Esta sección describe los problemas que comúnmente ocurren cuando se crean o se usan los datos URIs.

Sintaxis

El formato de los datos URIs es muy simple, pero es facil olvidarse de poner una coma antes del segmento de la "data", o para codificar incorrectamente los datos en formato base64.

Formateando en HTML

Un dato URI provee un archivo dentro de un archivo, que potencialmente puede ser muy amplia con relación con el ancho del documento de cierre. Como una URL, los datos se les puede dar formato con espacios en blanco (avance de línea, pestaña, o espacios), pero hay cuestiones prácticas que se plantean cuando se usa codificación base64.

Limitaciones de longitud

Aunque Firefox soporta con URIs de datos de longitud esencialmente ilimitada, los navegadores no están obligados a apoyar cualquier longitud máxima de datos en particular. Por ejemplo, el navegador Opera 11 limita las URIs de datos cerca de `los` 65000 caracteres.

Falta de control de errores

Los parámetros no válidos en los medios de comunicación, o errores ortográficos cuando se especifiquen `'base64'`, se ignoran, pero no se proporciona ningún error.

No hay soporte para consulta de cadenas, etc.

Las partes de datos de URIs de datos son opacos, por lo que un intento de utilizar una cadena de consulta (parámetros específicos de página, con la sintaxis `<url>?parameter-data`) con un URIs de datos que se acaba de incluir la cadena de consulta en los datos de la URI que representa. Por ejemplo:

```
data:text/html,lots of text…<p><a
name%3D”bottom”>bottom</a>?arg=val
```

Esto representa un recurso HTML cuyo contenido es:

```
lots of text…<p><a name=”bottom”>bottom</a>?arg=val
```

Especificaciones

Especificación	Título
RFC 2397	The "data" URL scheme"

Tipos MIME

El **tipo Extensiones multipropósito de Correo de Internet (MIME)** es una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos. Está definido y estandarizado en IETF RFC 6838.

La Autoridad de Números Asignados de Internet (IANA) es el organismo oficial responsable de realizar un seguimiento de todos los tipos MIME oficiales, y puede encontrar la lista más actualizada y completa en la página de tipos de medios (Media Types).

Los navegadores a menudo usan el tipo MIME (y no la extensión de archivo) para determinar cómo procesará un documento; por lo tanto, es importante que los servidores estén configurados correctamente para adjuntar el tipo MIME correcto al encabezado del objeto de respuesta.

Sintaxis

Estructura general

```
tipo/subtipo
```

La estructura de un tipo MIME es muy simple; consiste en un tipo y un subtipo, dos cadenas, separadas por un `'/'`. No se permite espacio. El *tipo* representa la categoría y puede ser de tipo *discreto* o *multiparte*. El *subtipo* es específico para cada tipo.

Un tipo MIME no distingue entre mayúsculas y minúsculas, pero tradicionalmente se escribe todo en minúsculas.

Tipos discretos

```
text/plain

text/html

image/jpeg

image/png

audio/mpeg

audio/ogg

audio/*

video/mp4

application/octet-stream

…
```

Los tipos *discretos* indican la categoría del documento, puede ser uno de los siguientes:

Tipo	Descripción	Ejemplo de subtipos típicos
`text`	Representa cualquier documento que contenga texto y es teóricamente legible por humanos	`text/plain`, `text/html`, `text/css`, `text/javascript`
`image`	Representa cualquier tipo de imagen. Los videos no están incluidos, aunque las imágenes anima	`image/gif`, `image/png`, `image/jpeg`, `image/bmp`, `image/webp`

Tipo	Descripción	Ejemplo de subtipos típicos
	das (como el gif animado) se describen con un tipo de imagen.	
`audio`	Representa cualquier tipo de archivos de audio	`audio/midi`, `audio/mpeg`, `audio/webm`, `audio/ogg`, `audio/wav`
`video`	Representa cualquier tipo de archivos de video	`video/webm`, `video/ogg`

Tipo	Descripción	Ejemplo de subtipos típicos
`application`	Representa cualquier tipo de datos binarios.	`application/octet-stream`, `application/pkcs12`, `application/vnd.mspowerpoint`, `application/xhtml+xml`, `application/xml`, `application/pdf`

Para documentos de texto sin subtipo específico, se debe usar `text/plain`. De forma similar, para los documentos binarios sin subtipo específico o conocido, se debe usar `application/octet-stream`.

Tipos multiparte

```
multipart/form-data

multipart/byteranges
```

Los tipos de *partes múltiples* indican una categoría de documento que está rota en distintas partes, a menudo con diferentes tipos de MIME. Es una forma de representar un documento compuesto. Con la excepción de `multipart/form-data`, que se utilizan en relación con formularios HTML y el método `POST`, y `multipart/byteranges`, que se utilizan junto con el mensaje de estado de `Contenido Parcial 206` para enviar solo un subconjunto de un documento completo, HTTP no maneja documentos multiparte de una manera específica: el mensaje simplemente se transmite al navegador (que probablemente propondrá una ventana Guardar como, sin saber cómo mostrar el documento en línea.)

Tipos MIME importantes para desarrolladores Web

`application/octet-stream`

Este es el valor predeterminado para un archivo binario. Como realmente significa un *archivo binario desconocido*, los navegadores generalmente no lo

ejecutan automáticamente, o incluso preguntan si debería ejecutarse. Lo tratan como si el encabezado `Content-Disposition` se configurara con el valor `attachment` y proponen un archivo 'Guardar como'.

`text/plain`

Este es el valor predeterminado para los archivos de texto. Incluso si realmente significa un archivo textual desconocido, los navegadores asumen que pueden mostrarlo.

Tenga en cuenta que `text/plain` no significa *cualquier tipo de datos textuales*. Si esperan un tipo específico de datos textuales, probablemente no lo considerarán una coincidencia. Específicamente, si descargan un archivo de texto sin formato `text/plain` de un elemento `<link>` que declara archivos CSS, no lo reconocerán como un archivo CSS válido si se presenta con `text/plain`. Se debe usar el tipo MIME CSS `text/css`.

`text/css`

Todos los archivos CSS que deban interpretarse como tales en una página web **deben** ser de los archivos de `text/css`. A menudo, los servidores no reconocen los archivos con el sufijo `.css` como archivos CSS y los envían con tipo MIME `text/plain` o `application/octet-stream`: en estos casos, la mayoría de los navegadores no los reconocerán como archivos CSS y serán ignorado silenciosamente. Se debe prestar especial atención en servir los archivos CSS con el tipo correcto.

`text/html`

Todo el contenido HTML debe ser servido con este tipo. Los tipos MIME alternativos para XHTML, como `application/xml+html`, son en su mayoría inútiles hoy en día (HTML5 unificó estos formatos).

Tipos de imágenes

Solo un puñado de tipos de imágenes son ampliamente reconocidos y se consideran seguros para la Web, listos para usar en una página Web:

Tipo MIME	Tipo de imagen
`image/gif`	Imágenes GIF (compresión sin pérdida, reemplazada por PNG)

Tipo MIME	Tipo de imagen
`image/jpeg`	Imágenes JPEG
`image/png`	Imágenes PNG
`image/svg+xml`	Imágenes SVG (imágenes vectoriales)

Existe una discusión para agregar WebP (`image/webp`) a esta lista, pero como cada tipo de imagen nuevo aumentará el tamaño de una base de código, esto puede presentar nuevos problemas de seguridad, por lo que los proveedores de navegadores son cautelosos al aceptarlo.

Se pueden encontrar otros tipos de imágenes en documentos Web. Por ejemplo, muchos navegadores admiten tipos de imágenes de iconos para favicones o similares. En particular, las imágenes ICO son compatibles en este contexto con el tipo MIME `image/x-icon`.

Tipos de audio y video

Al igual que las imágenes, HTML no define un conjunto de tipos soportados para usar con los elementos `<audio>` y `<video>`, por lo que solo un grupo relativamente pequeño de ellos puede ser utilizado en la web. Los formatos de medios compatibles con los elementos de audio y video HTML explican tanto los códecs como los formatos de contenedor que se pueden usar.

El tipo MIME de dichos archivos representa principalmente los formatos de contenedor y los más comunes en un contexto web son:

Tipo MIME	Tipo de audio o video
`audio/wave` `audio/wav` `audio/x-wav` `audio/x-pn-wav`	Un archivo de audio en formato de contenedor WAVE. El códec de audio PCM (códec WAVE "1") a menudo es compatible, pero otros códecs tienen soporte más limitado (si lo hay).
`audio/webm`	Un archivo de audio en formato de contenedor WebM. Vorbis y Opus son los códecs de audio más comunes.
`video/webm`	Un archivo de video, posiblemente con audio, en el formato de contenedor de WebM. VP8 y VP9 son los códecs de video más comunes utilizados en él; Vorbis y Opus los códecs de audio más comunes.
`audio/ogg`	Un archivo de audio en el formato de contenedor OGG. Vorbis es el códec de audio más común utilizado en dicho contenedor.
`video/ogg`	Un archivo de video, posiblemente con audio, en el formato de contenedor OGG. Theora es el códec de video habitual utilizado en él; Vorbis es el códec de audio habitual.
`application/ogg`	Un archivo de audio o video usando el formato de contenedor OGG. Theora es el códec de video habitual utilizado en él; Vorbis es el códec de audio más común.

`multipart/form-data`

El tipo de datos `multipart/form-data` se puede usar al enviar el contenido de un formulario HTML completo desde el navegador al servidor. Como formato de documento multiparte, consta de diferentes partes, delimitadas por un límite (una cadena que comienza con un doble guion `'--'`). Cada parte es una entidad en sí misma, con sus propios encabezados HTTP, `Content-Disposition` y `Content-Type` para los campos de carga de archivos, y los más comunes (`Content-Length` es ignorado ya que la línea límite se usa como delimitador).

```
Content-Type: multipart/form-data;
boundary=unaCadenaDelimitadora

(otros encabezados asociados con el documento multiparte como un
todo)



--unaCadenaDelimitadora

Content-Disposition: form-data; name="miArchivo";
filename="img.jpg"

Content-Type: image/jpeg



(data)

--unaCadenaDelimitadora

Content-Disposition: form-data; name="miCampo"



(data)

--unaCadenaDelimitadora

(más subpartes)

--unaCadenaDelimitadora--
```

El siguiente formulario:

```
<form action="http://localhost:8000/" method="post"
enctype="multipart/form-data">

  <input type="text" name="miCampoDeTexto">

  <input type="checkbox" name="miCheckBox">Checado</input>

  <input type="file" name="miArchivo">

  <button>Enviar el archivo</button>

</form>
```

enviará este mensaje:

```
POST / HTTP/1.1

Host: localhost:8000

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:50.0) Gecko/20100101 Firefox/50.0

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: multipart/form-data; boundary=--------------------
-------872165604191141565395500449 8

Content-Length: 465



-----------------------------872165604191141565395500449 8

Content-Disposition: form-data; name="miCampoDeTexto"
```

```
Test

-----------------------------872165604191141565395500449 8

Content-Disposition: form-data; name="miCheckBox"



on

-----------------------------872165604191141565395500449 8

Content-Disposition: form-data; name="miArchivo";
filename="prueba.txt"

Content-Type: text/plain



Simple file.

-----------------------------872165604191141565395500449 8--
```

`multipart/byteranges`

El tipo MIME `multipart/byteranges` se usa en el contexto del envío de respuestas parciales al navegador. Cuando se envía el código de estado de `Contenido Parcial` `206`, este tipo MIME se usa para indicar que el documento está compuesto de varias partes, una para cada rango solicitado. Al igual que otros tipos de varias partes, `Content-Type` usa la directiva `boundary` para definir la cadena delimitadora. Cada una de las diferentes partes tiene un encabezado `Content-Type` con el tipo real del documento y con el rango que representan.

```
HTTP/1.1 206 Partial Content

Accept-Ranges: bytes

Content-Type: multipart/byteranges; boundary=3d6b6a416f9b5

Content-Length: 385



--3d6b6a416f9b5
```

```
Content-Type: text/html

Content-Range: bytes 100-200/1270



eta http-equiv="Content-type" content="text/html; charset=utf-8" />

    <meta name="vieport" content

--3d6b6a416f9b5

Content-Type: text/html

Content-Range: bytes 300-400/1270



-color: #f0f0f2;

        margin: 0;

        padding: 0;

        font-family: "Open Sans", "Helvetica

--3d6b6a416f9b5--
```

Importancia de establecer el tipo MIME correcto

La mayoría de los servidores web envían recursos de tipo desconocido utilizando el tipo MIME predeterminado `application/octet-stream`. Por razones de seguridad, la mayoría de los navegadores no permiten establecer una acción predeterminada personalizada para dichos recursos, lo que obliga al usuario a almacenarlo en el disco para usarlo. Algunas configuraciones de servidor incorrectas que se ven con frecuencia ocurren con los siguientes tipos de archivos:

- Archivos con codificación RAR. En este caso, lo ideal sería establecer el tipo verdadero de los archivos codificados; esto a menudo no es posible (ya que puede no ser conocido por el servidor y estos archivos pueden contener varios recursos de diferentes tipos). En este caso, al configurar el servidor para que envíe el tipo MIME `application/x-rar-compressed`, los usuarios no habrán definido una acción predeterminada útil para ellos.

- Archivos de audio y video. Solo los recursos con el tipo MIME correcto serán reconocidos y reproducidos en elementos `<video>` o `<audio>`. Asegúrese de usar el tipo correcto para audio y video.

- Tipos de archivos propietarios. Preste especial atención al servir un tipo de archivo propietario. Evite el uso de `application/octet-stream` ya que no será posible un manejo especial: la mayoría de los navegadores no permiten definir un comportamiento predeterminado (como "Abrir en Word") para este tipo genérico MIME.

Olfateo MIME (sniffing)

En ausencia de un tipo MIME, o en algunos otros casos en los que un cliente cree que están configurados incorrectamente, los navegadores pueden realizar el rastreo MIME, que es adivinar el tipo MIME correcto mirando el recurso. Cada navegador realiza esto de manera diferente y bajo diferentes circunstancias. Hay algunas preocupaciones de seguridad con esta práctica, ya que algunos tipos MIME representan el contenido ejecutable y otros no. Los servidores pueden bloquear el rastreo de MIME enviando el `X-Content-Type-Options` a lo largo de `Content-Type`.

Otros métodos de transporte de tipo de documento

Los tipos MIME no son la única forma de transmitir la información del tipo de documento:

- Los sufijos de nombre a veces se usan, especialmente en los sistemas de Microsoft Windows. No todos los sistemas operativos consideran significativos estos sufijos (especialmente Linux y Mac OS), y al igual que un tipo MIME externo, no hay garantía de que sean correctos.
- Números mágicos. La sintaxis de los diferentes tipos de archivos permite la inferencia del tipo de archivo al observar la estructura. P.ej. cada archivo GIF comienza con el valor hexadecimal 47 49 46 38 39 [GIF89] o archivos PNG con 89 50 4E 47 [.PNG]. No todos los tipos de archivos tienen números mágicos, por lo que este tampoco es un sistema 100% confiable.

Lista completa de tipos MIME

Aquí está una lista completa de tipos de MIME, asociados por tipo de documentos y ordenados por su extensión común.

Dos tipos primarios de MIME son importantes para el rol de tipos por defecto:

- `text/plain` es el valor por defecto para los archivos textuales. Un archivo textual debe ser legible y no puede contener datos binarios.
- `application/octet-stream` es el valor por defecto para todos los demás casos. Un tipo de archivo desconocido debe usar este tipo. Los navegadores tienen un cuidado particular cuando manipulan estos archivos, tratando de proteger al usuario previéndo comportamientos peligrosos.

IANA es el registro oficial de los tipos de media MIME y mantiene una lista oficial de todos los tipos de MIME. Esta tabla, lista algunos de los tipos de MIME importantes para la web:

Extensión	Tipo de documento	Tipo de MIME
`.aac`	Archivo de audio AAC	`audio/aac`
`.abw`	Documento Abi Word	`application/x-abiword`
`.arc`	Documento de Archivo (múltiples archivos incrustados)	`application/octet-stream`
`.avi`	AVI: Audio Video Intercalado	`video/x-msvideo`
`.azw`	Formato eBook Amazon Kindle	`application/vnd.amazon.ebook`

Extensión	Tipo de documento	Tipo de MIME
`.bin`	Cualquier tipo de datos binarios	`application/octet-stream`
`.bz`	Archivo BZip	`application/x-bzip`
`.bz2`	Archivo BZip2	`application/x-bzip2`
`.csh`	Script C-Shell	`application/x-csh`
`.css`	Hojas de estilo (CSS)	`text/css`
`.csv`	Valores separados por coma (CSV)	`text/csv`
`.doc`	Microsoft Word	`application/msword`

Extensión	Tipo de documento	Tipo de MIME
`.epub`	Publicación Electrónica (EPUB)	`application/epub+zip`
`.gif`	Graphics Interchange Format (GIF)	`image/gif`
`.htm` `.html`	Hipertexto (HTML)	`text/html`
`.ico`	Formato Icon	`image/x-icon`
`.ics`	Formato iCalendar	`text/calendar`
`.jar`	Archivo Java (JAR)	`application/java-archive`
`.jpeg` `.jpg`	Imágenes JPEG	`image/jpeg`

Extensión	Tipo de documento	Tipo de MIME
`.js`	JavaScript (ECMAScript)	`application/javascript`
`.json`	Formato JSON	`application/json`
`.mid` `.midi`	Interfaz Digital de Instrumentos Musicales (MIDI)	`audio/midi`
`.mpeg`	Video MPEG	`video/mpeg`
`.mpkg`	Paquete de instalación de Apple	`application/vnd.apple.installer+xml`
`.odp`	Documento de presentación de OpenDocument	`application/vnd.oasis.opendocument.presentation`
`.ods`	Hoja de Cálculo OpenDocument	`application/vnd.oasis.opendocument.spreadsheet`

Extensión	Tipo de documento	Tipo de MIME
`.odt`	Documento de texto OpenDocument	`application/vnd.oasis.opendocument.text`
`.oga`	Audio OGG	`audio/ogg`
`.ogv`	Video OGG	`video/ogg`
`.ogx`	OGG	`application/ogg`
`.pdf`	Adobe Portable Document Format (PDF)	`application/pdf`
`.ppt`	Microsoft PowerPoint	`application/vnd.ms-powerpoint`
`.rar`	Archivo RAR	`application/x-rar-compressed`

Extensión	Tipo de documento	Tipo de MIME
`.rtf`	Formato de Texto Enriquecido (RTF)	`application/rtf`
`.sh`	Script Bourne shell	`application/x-sh`
`.svg`	Gráficos Vectoriales (SVG)	`image/svg+xml`
`.swf`	Small web format (SWF) o Documento Adobe Flash	`application/x-shockwave-flash`
`.tar`	Aerchivo Tape (TAR)	`application/x-tar`
`.tif` `.tiff`	Formato de archivo de imagen etiquetado (TIFF)	`image/tiff`

Extensión	Tipo de documento	Tipo de MIME
`.ttf`	Fuente TrueType	`font/ttf`
`.vsd`	Microsft Visio	`application/vnd.visio`
`.wav`	Formato de audio de forma de onda (WAV)	`audio/x-wav`
`.weba`	Audio WEBM	`audio/webm`
`.webm`	Video WEBM	`video/webm`
`.webp`	Imágenes WEBP	`image/webp`
`.woff`	Formato de fuente abierta web (WOFF)	`font/woff`

Extensión	Tipo de documento	Tipo de MIME
`.woff2`	Formato de fuente abierta web (WOFF)	`font/woff2`
`.xhtml`	XHTML	`application/xhtml+xml`
`.xls`	Microsoft Excel	`application/vnd.ms-excel`
`.xml`	`XML`	`application/xml`
`.xul`	XUL	`application/vnd.mozilla.xul+xml`
`.zip`	Archivo ZIP	`application/zip`
`.3gp`	Contenedor de audio/video 3GPP	`video/3gpp` `audio/3gpp` if it doesn't contain video

Extensión	Tipo de documento	Tipo de MIME
`.3g2`	Contenedor de audio/video 3GP P2	`video/3gpp2` `audio/3gpp2` if it doesn't contain video
`.7z`	Archivo 7-zip	`application/x-7z-compressed`

Configurar correctamente los tipos MIME del servidor

Introduccion

Por omisión, muchos servidores web estan configurados para reportar un tipo MIME de `texto/plano` ó `aplicacion/de fuente de octeto` para tipos de contenidos desconocidos. A medida son desarrollados nuevos tipos de contenidos, los administradores de red pueden equivocarse al añadirlos a la configuración del servidor web, y esta es la principal causa de problemas para usuarios de navegadores basados en Gecko, el cual respeta los tipos MIME tal y como son reportados por los servidores y las aplicaciones web.

¿Que son los tipos de MIME?

Los tipos de MIME describen el tipo de medio del contenido, sea del correo electrónico o el utilizado en los servidores o aplicaciones web, y tiene como propósito ayudar a guiar al navegador web acerca de cómo ha de ser procesado y mostrado el contenido. Ejemplos de tipos de MIME son:

- `texto/html` para páginas web normales
- `texto/plano` para texto común
- `Aplicacion/de fuente octeto` que significa "descarga este archivo"
- `Aplicacion/x-java-applet` para uso de applets de Java

- `Aplicacion/pdf` para documentos de Adobe® PDF.

Información Técnica

MIME está actualmente definida en RFCs 2045, 2046, 2047, 2048, y 2049 y los valores registrados para los tipos MIME estan disponibles en IANA | MIME Media Types. La HTTP specification define un superconjunto de MIME el cual es utilizado para describir los tipos de medios usados en la red.

¿Por qué son importantes los tipos correctos de MIME?

Si el servidor de red o la aplicación informan un tipo incorrecto de MIME para el contenido, un navegador de red no tiene forma de saberlo, *de acuerdo con la especificación HTTP*, más si tenemos en cuenta que el autor especificó el contenido para ser procesado y mostrado en una forma diferente que la impuesta por el tipo MIME informado.

Otros navegadores de red, tal como el Microsoft® Internet Explorer, intentan determinar el tipo adecuado de MIME en servidores mal configurados, suponiendo el tipo adecuado de MIME que debería ser. Esto protege a muchos administradores de red de sus propios errores, pues el Internet Explorer continúa procesando el contenido, aunque, por ejemplo, una imagen haya sido informada como texto plano.

¿Por qué los navegadores no deberían suponer tipos MIME?

A parte de la violación de la especificación HTTP, es una mala estrategia para los navegadores suponer tipos MIME por las siguientes razones:

Pérdida del control

Si el navegador ignora el tipo MIME reportado, los administradores de red y los autores dejarán de tener el control sobre cómo sus contenidos serán procesados.

Por ejemplo, un sitio de red orientado para desarrolladores de red puede enviar determinados ejemplos de documentos HTML como enteros `text/html` ó como `text/plain` para lograr documentos con enteros procesados y mostrados como HTML ó como código fuente. Si el navegador supone el tipo MIME, esta posibilidad dejará de estar disponible para el autor.

Seguridad

Algunos tipos de contenidos, tales como programas ejecutables, son inherentemente inseguros. Por este motivo, esos tipos MIME son generalmente restringidos en términos de qué acciones tomará el navegador de red al recibirlos. Por ejemplo, un programa ejecutable no debería ser ejecutado en la

computadora de un usuario, y en su lugar debería aparecer un cuadro de diálogo para preguntar al usuario si desea descargar el archivo.

La suposición de tipos MIME ha llevado a fallas de seguridad en Internet Explorer, debido a autores maliciosos que reportaban el tipo MIME de un archivo peligroso como si fuera uno seguro, evitando así el cuadro de diálogo de descarga normal. El Internet Explorer suponía entonces que se trataba de un programa ejecutable y los corría en la computadora del usuario.

Cómo determinar el tipo MIME enviado a un servidor

En Firefox, cargar el archivo y usar Herramientas | Información de página. Puede también usar Rex Swain's HTTP Viewer ó Live HTTP Headers para ver los encabezados completos y contenido de cualquier archivo enviado desde un servidor de red.

De acuerdo con los estándares, una `meta` etiqueta que brinda el tipo MIME tal como `<meta http-equiv="Content-Type" content="text/html">` debería ser ignorado si hay una `Content-Type` línea en el encabezado. En vez de buscar esta línea en la fuente HTML, use las técnicas anteriores para determinar el tipo MIME enviado por un servidor.

Cómo determinar el tipo correcto de MIME para su contenido

Hay ciertos pasos los cuales pueden llevar a determinar el correcto tipo de valor MIME a ser usado para su contenido.

1. Si su contenido fue creado usando la aplicación de software del vendedor, lea la documentación del vendedor para ver qué tipos MIME debería n ser informados para los diferentes tipos de medios.
2. Mire en el IANA | Registro de Tipos de medios MIME que contiene todos los tipos MIME registrados.
3. Si el tipo de medio es mostrado usando un plug-in en Netscape Gecko, instale el plug-in y luego mire en Ayuda->Acerca en le Menú Plug-in para ver qué tipos MIME están asociados con el tipo de medio.
4. Buscar la extensión del archivo en FILExt para ver qué tipos MIME están asociados con esa expresión.

Cómo configurar su servidor para enviar los tipos MIME correctos

- Si está usando un servidor de red Apache, simplemente copie esto sample .htaccess file en el directorio que contiene los archivos que quiere enviar con los tipos correctos MIME. Si tiene un subdirectorio entero de archivos, sólo ubique el archivo en el directorio principal; No necesita ubicarlo en cada subdirectorio.
- Si usa Microsoft IIS, vea este artículo en Microsoft TechNet.

- Si utiliza un servidor script para generar contenido, puede generalmente agregar una línea cerca del principio de su script. Puede servir contenido otro que HTML desde Perl, PHP, ASP, ó Java — sólo cambie el tipo MIME adecuado.

    - Para Perl CGI, debería tener la línea `print "Content-Type: text/html\n\n";` antes que cualesquiera otras líneas de salida. Si utiliza el módulo CGI, puede utilizar la línea `print $cgi->header('text/html');` en lugar de, donde `$cgi` es su referencia para la instancia CGI.
    - Para PHP, debería tener la línea `header('Content-Type: text/html');` antes que cualesquiera otras líneas de salida.
    - Para ASP, debería tener la línea `response.ContentType = "text/html";` antes que cualesquiera otras líneas de salida.
    - Para un servlet Java, debería tener la línea `response.setContentType("text/html");` al principio de su `doGet` ó `doPost` método, donde `response` es una referencia a `HttpServletResponse`.
-

Enlaces Relacionados

- IANA | MIME Media Types
- Hypertext Transfer Protocol — HTTP/1.0
- Microsoft - 293336 - INFO: WebCast: MIME Type Handling in Microsoft Internet Explorer
- Microsoft - Appendix A: MIME Type Detection in Internet Explorer
- Microsoft - Security Update, March 29, 2001
- Microsoft - Security Update, December 13, 2001

Elección entre www y no-www URLs

Una cuestión recurrente entre los dueños de sitios web consiste en elegir entre un no-www y www URLs. Esta página contiene algunos consejos sobre qué es mejor.

¿Qué son los nombres de dominio?

En una URL HTTP, la primera subcadena que sigue a la inicial `http://` o `https://` se llama nombre de dominio. El nombre de dominio está alojado en un servidor donde residen los documentos.

Un servidor no es necesariamente una máquina física: varios servidores pueden residir en la misma máquina física. O bien, un servidor puede ser manejado por varias máquinas, cooperando para producir la respuesta o equilibrando la carga de las solicitudes entre ellas. El punto clave es que semánticamente un nombre de dominio representa un solo servidor.

¿Entonces, ¿tengo que elegir uno u otro para mi sitio web?

- Sí, tienes que elegir uno y seguir con él. La elección de cual tener como ubicación canónica es tuya, pero si eliges una, síguela. Esto hará tu sitio web parezca más consistente para tus usuarios y para los motores de búsqueda. Esto incluye siempre vincular al dominio elegido (lo que no debería ser difícil si está utilizando URLs relativas en su sitio web) y siempre compartir enlaces (por correo electrónico / redes sociales, etc.) al mismo dominio.
- No, puedes tener dos. Lo que es importante es que seas coherente y consistente con cuál es el dominio oficial. **A este dominio oficial se le llama nombre *canónico*.** Todos tus enlaces absolutos deben usarlo. Pero, aun así, puedes seguir teniendo el otro dominio funcionando: HTTP permite dos técnicas para que esté claro para sus usuarios, o motores de búsqueda, cuyo dominio es el canónico, mientras permite que el dominio no-canónico funcione para proporcionar las páginas esperadas.

Por lo tanto, ¡elija uno de sus dominios como su canónico! Hay dos técnicas a continuación para permitir que el dominio no canónico funcione aún.

Técnicas para las URL canónicas.

Hay diferentes maneras de elegir qué sitio web es *canónico*.

Usando redirecciones HTTP 301

En este caso, necesitas configurar el servidor que recibe las peticiones HTTP (que probablemente sea el mismo para las URL www y no www) para responder con una respuesta HTTP adecuada 301 a cualquier solicitud al dominio no-canónico. Esto redirigirá el navegador que intenta acceder a las URL no canónicas a su equivalente canónico. Por ejemplo, si ha elegido usar URL que no sean de www como tipo canónico, debe redirigir todas las URL de www a su URL equivalente sin el www.

Ejemplo:

1. Un servidor recibe una petición para `http://www.example.org/whaddup` (cuando el dominio canónico es example.org)
2. El servidor responde con un código 301 con la cabecera `Location` `http://example.org/whaddup`.
3. El cliente emite una solicitud al dominio canónico.: `http://example.org/whatddup`

El HTML5 boilerplate project tiene un ejemplo sobre cómo configurar un servidor Apache para redirigir un dominio a otro.

Usando `<link rel="canonical">`

Es posible añadir un elemento especial HTML `<link>` a una página para indicar cual es la dirección canónica de la página. Esto no tendrá ningún impacto en la lectura humana, pero le dirá a rastreadores de los motores de búsqueda donde vive actualmente la página. De esta manera, los motores de búsqueda no indexan la misma página varias veces, lo que podría hacer que se considere contenido duplicado o correo no deseado, e incluso eliminar o bajar su página de las páginas de resultados del motor de búsqueda.

Al agregar una etiqueta de este tipo, sirve el mismo contenido para ambos dominios, indicando a los motores de búsqueda qué URL es canónica. En el ejemplo anterior, `http://www.example.org/whaddup` serviría el mismo contenido que http://example.org/whaddup, pero con un elemento `<link>` adicional en la cabecera:

```
<link href="http://example.org/whaddup" rel="canonical">
```

A diferencia del caso anterior, el historial del navegador considerará las direcciones URL no www y www como entradas independientes.

Hacer que tu página funcione para ambas

Con estas técnicas, puedes configurar tu servidor para responder correctamente a ambos dominios, www y no www. Es un buen consejo hacer esto, ya que no puede predecir qué URL escribirán los usuarios en la barra de URL de su navegador. Es una cuestión de elegir qué tipo desea usar como su ubicación canónica, y luego redirigir el otro tipo a él.

Decidir el caso

Este es un tema muy subjetivo que podría considerarse un problema de bikeshedding. Si desea leer más a fondo, lea algunos de los muchos artículos de este tema.

Ver también

- Estadísticas sobre lo que la gente escribe en la barra de URL (2011)

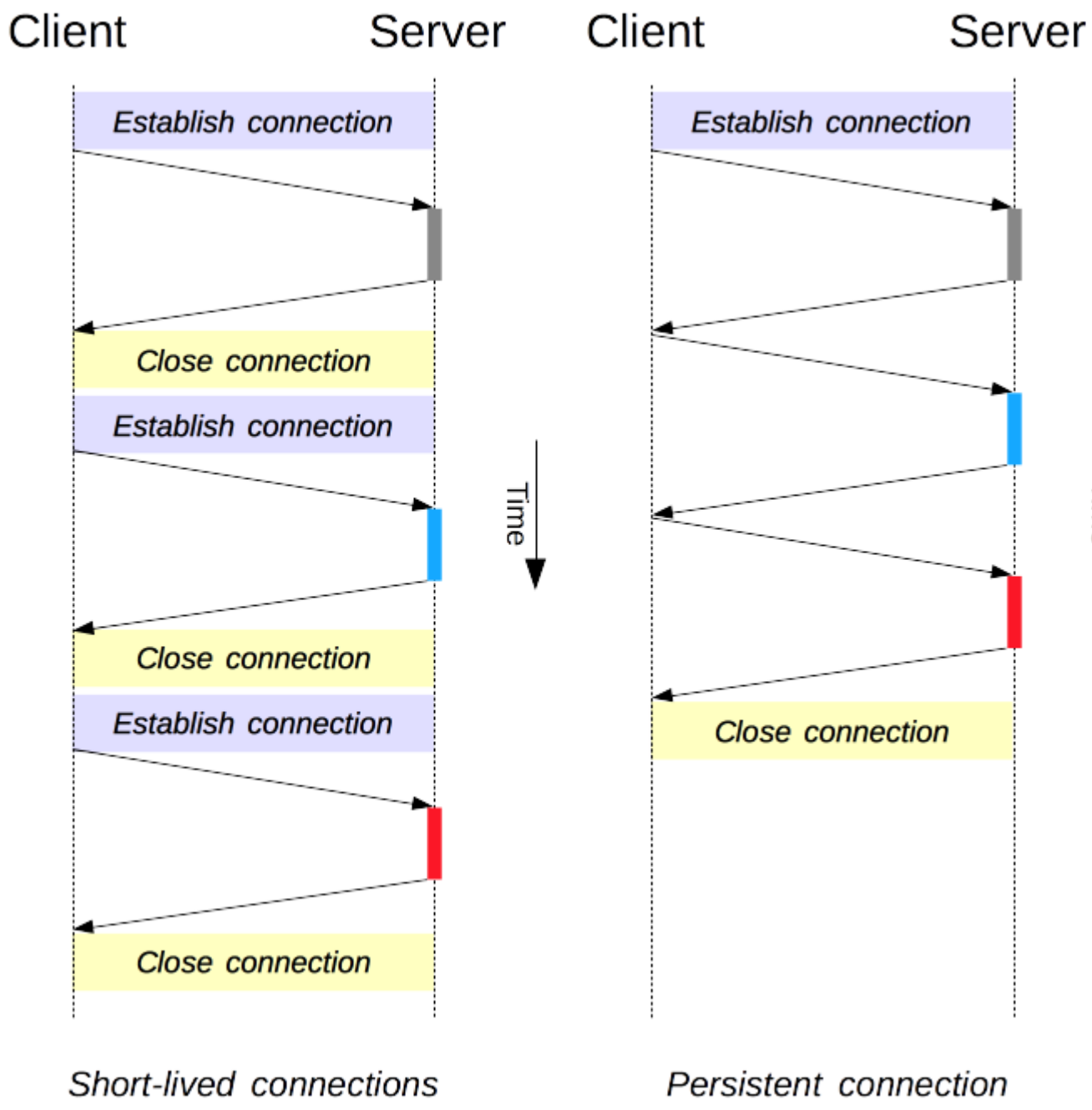
Gestión de la conexión en HTTP/1.x

La gestión de las conexiones en un tema fundamental en HTTP: crear y mantener una conexión tiene una gran influencia en el rendimiento de las páginas Web y las aplicaciones Web. En la versión HTTP/1.x, hay modos de conexión: conexiones breves, conexiones persistentes, y '*pipelining*'.

HTTP la mayor parte de las veces se basa en TCP, como protocolo de transporte, dando conexión entre el cliente y el servidor. En sus comienzos, HTTP, únicamente tenia un modelo para gestionar dichas conexiones. Aquellas conexiones eran conexiones breves: se creaba una cada vez que una petición necesitaba ser transmitida, y se cerraba, una vez que la respuesta se había recibido.

Este sencillo modelo tenía una limitación intrínseca en su rendimiento: abrir una conexión TCP es una operación costosa en recursos. Se necesitan intercambiar varios mensajes entre el cliente y el servidor para hacerlo. Y la latencia de la red y su ancho de banda, se veían afectados cuando se realizaba una petición. Las páginas web actuales, necesitan varias peticiones (una docena o más) para tener la información necesaria, de manera que este primer modelo es ineficiente para ellas.

Dos nuevos modelos se presentaron en HTTP/1.1. La conexión persistente, mantiene las conexiones abiertas, entre peticiones sucesivas, eliminando así el tiempo necesario para abrir nuevas conexiones. El modelo 'pipelining' va un paso más allá, y envía varias peticiones sucesivas, sin esperar por la respuesta, reduciendo significativamente la latencia en la red.



HTTP/2 añade nuevos modelos para la gestión de la conexión.

Un punto significativo a tener en cuenta en la gestión de la conexión de HTTP, es que este se refiere a la conexión establecida entre dos nodos consecutivos de la red, esto se denomina hop-by-hop, en contraposición al concepto de end-to-end. El modelo de conexión entre un cliente y su primer proxy, puede ser distinto que la comunicación entre el proxy y el servidor de destino (u otro proxy intermedio). Las cabeceras de HTTP utilizadas para definir el modelo de conexión como Connection (en-US) y `Keep-Alive`, se refieren a una conexión hop-by-hop, y estos parámetros, pueden variar en las comunicaciones de los nodos intermedios.

Un tema también relativo a esto, es el concepto de conexiones con protocolos HTTP superiores, donde una conexión HTTP/1.1 se actualiza a un protocolo distinto, como puede ser TLS/1.0, WebSockets, o HTTP/2. Este mecanismo de actualización del protocolo se encuentra documentado en otra página.

Conexiones breves

El modelo original de HTTP, y el modelo de HTTP/1.0, está basado, en conexiones breves. Cada petición HTTP, se completa estableciendo (iniciando, estableciendo y cerrando) su propia conexión. Esto supone que la coordinación en el protocolo HTTP (handshake), sucede de forma previa a cada petición HTTP.

La coordinación o inicialización de una comunicación en el protocolo TCP, requiere un tiempo dado, pero al adaptarse el protocolo TCP a su carga de transmisión de datos, este incrementa su eficiencia cuando se mantiene la conexión en el tiempo, utilizándose una conexión para transmitir numerosos peticiones de datos. Las conexiones breves, no utilizan esta característica del protocolo TCP, y el rendimiento de la conexión es peor que en el caso óptimo, al estar constantemente necesitando iniciar conexiones para transmitir cada mensaje (esto se conoce como conexiones 'en frio', o en inglés: 'cold connections').

La conexión breve es la conexión usada por defecto en HTTP/1.0 (únicamente en el caso de no esté definida la cabecera Connection (en-US), o su valor sea `close` entonces, no se utilizaria el modelo de conexiones breves). En HTTP/1.1, este modelo de conexión unicamente se utiliza al definir la cabecera Connection (en-US) como `close` .

A menos que se de la situación en que se ha de trabajar con sistemas antiguos que no soportan conexiones persistentes, no hay otra razón para el uso de este modelo de conexiones.

Conexiones persistentes

Las conexiones breves, tienen dos importantes desventajas: el tiempo que necesitan para establecer una nueva conexión es significativo, y la eficacia del protocolo inferior a HTTP: el TCP unicamente mejora cuando la conexión ha estado funcionando durante algún tiempo (conexión caliente o 'warm connection' en inglés). Para atenuar dichos inconvenientes, se presentó el concepto de conexión persistente, ya incluso antes de presentar el protocolo HTTP/1.1. También se le conoce como *'keep-alive connection*' que en inglés indica una conexión que se mantiene viva.

Una conexión persistente es aquella la cual permanece abierta por un periodo, y puede ser reutilizada por varias peticiones de datos, ahorrando así la necesidad de efectuar nuevas sincronizaciones a nivel de TCP, de esta manera se puede usar más eficazmente el protocolo TCP. Esta conexión no estará abierta

permanentemente, si la conexión no se usa, se cerrará después de un tiempo determinado (un servidor puede usar la cabecera `Keep-Alive` para definir el tiempo mínimo que la conexión debería estar abierta.

Las conexiones persistentes también tienen sus desventajas: incluso cuando no están transmitiendo datos, consumen recursos, y en casos en que la red esté soportando una carga de transferencia muy alta, un ataque DoS (en-US) puede realizarse. En estos casos, el uso de conexiones no persistentes, las cuales se cierran tan pronto como no necesitan transmitir, pueden resultar en un sistema más eficaz.

HTTP/1.0 las conexiones HTTP/1.0 no son persistentes por defecto. Si se indica en la cabecera de un mensaje Connection (en-US) cualquier otro valor que no sea: `close`, como puede ser: `retry-after`, en tal caso, se harán conexiones persistentes.

En HTTP/1.1 las conexiones son persistentes por defecto, así que esa cabecera no se necesita (aunque usualmente se añade como medida defensiva, en caso de que se tenga que utilizar el protocolo HTTP/1.0).

HTTP pipelining

HTTP pipelining no está activado por defecto en los navegacdores modernos:

- Proxies con defectos de implementación son habituales y provocan comportamientos extraños y erráticos, que los desarrolladores de Webs, no pueden predecir, ni corregir fácilmente.
- HTTP Pipelining es complicado de implementar correctamente: el tamaño del recurso pedido, el correcto RTT que será utilizado, así como el ancho de banda efectivo, tienen un impacto directo en la en la mejora de rendimiento de este método. Sin conocer estos valores, puede que mensajes importantes, se vean retrasados, por mensajes que no lo son. El concepto de "importante" incluso cambia según se carga la maquetación (layout) de la página. De ahí que este método solamente presente una mejora marginal en la mayoría de los casos.
- HTTP Pipelining presenta un problema conocido como HOL

Así, debido a estas razones este método ha sido relevado por un algoritmo mejor, la **multiplexación**, que es el que usa HTTP/2.

Por defecto, las peticiones HTTP son realizadas de manera sequencial. La siguiente petición es realizada una vez que la respuesta a la petición actual ha sido recibida. Debido a que se ven afectadas por latencias en la red y limitaciones en el ancho de banda, ésto puede llevar a retardos significativos hasta que la siguiente petición es *vista* por el servidor.

Pipelining es un proceso para enviar peticiones sucesivas, sobre la misma conexión persistente, sin esperar a la respuesta. Esto evita latencias en la conexión. En teoría, el rendimiento también podría mejorar si dos peticiones

HTTP se empaquetaran en el mismo mensaje TCP. El MSS (Maximum Segment Size) típico, es suficientemente grande para contener varias peticiones simples, aunque la demanda de tamaño de las peticiones HTTP sigue creciendo.

No todos los tipos de peticiones HTTP pueden ser utilizadas en pipeline: solo métodos idempotent (en-US) como `GET`, HEAD (en-US), `PUT` and DELETE (en-US) pueden ser repetidos sin incidencias. Si ocurre un fallo, el contenido se puede simplemente reenviar.

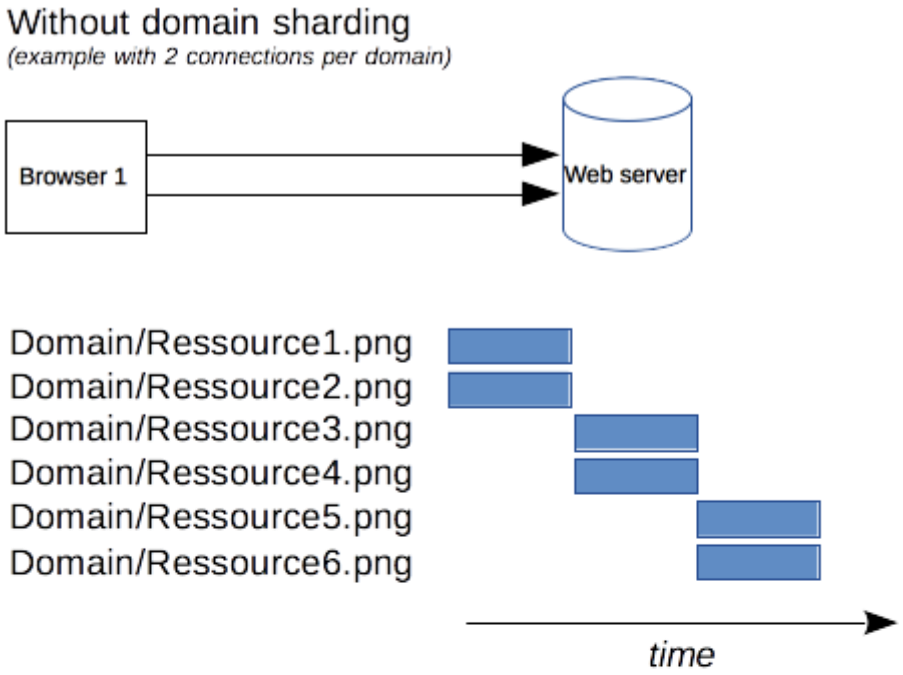
Hoy en día, todo proxy y servidor que cumpla con HTTP/1.1 debería dar soporte a pipelining, aunque en práctica tenga muchas limitaciones. Por esta razón, ningún explorador moderno lo activa por defecto.

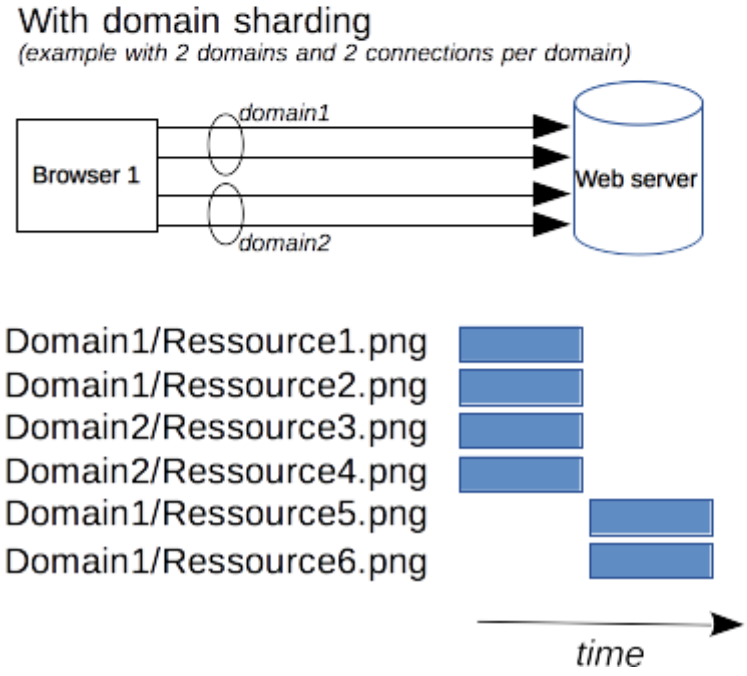
Domain sharding

Unless you have a very specific immediate need, don't use this deprecated technique; switch to HTTP/2 instead. In HTTP/2, domain sharding is no more useful: the HTTP/2 connection is able to handle parallel unprioritized requests very well. Domain sharding is even detrimental to performance. Most HTTP/2 implementation use a technique called connection coalescing to revert eventual domain sharding.

As an HTTP/1.x connection is serializing requests, even without any ordering, it can't be optimal without large enough available bandwidth. As a solution, browsers open several connections to each domain, sending parallel requests. Default was once 2 to 3 connections, but this has now increased to a more common use of 6 parallel connections. There is a risk of triggering DoS protection on the server side if attempting more than this number.

If the server wishes a faster Web site or application response, it is possible for the server to force the opening of more connections. For example, Instead of having all resources on the same domain, say `www.example.com`, it could split over several domains, `www1.example.com`, `www2.example.com`, `www3.example.com`. Each of these domains resolve to the *same* server, and the Web browser will open 6 connections to each (in our example, boosting the connections to 18). This technique is called *domain sharding*.





Conclusión

Las mejoras en la gestión de las conexiones, han significado un considerable aumento en el rendimiento del protocolo HTTP. Con HTTP/1.1 o HTTP/1.0, el uso de una conexión persistente - al menos hasta que que no necesite transmitir más datos- resulta en un significativo aumento en el rendimiento de la comunicación. Incluso, el fracaso de. método de pipelining, ha resultado en el diseño de modelos superiores para la gestión de la conexión, todo lo cual se ha incorporado en HTTP/2.