EBook Gratis

APRENDIZAJE HTTP

Free unaffiliated eBook created from
Stack Overflow contributors.

#http

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: http

It is an unofficial and free HTTP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official HTTP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con HTTP

Observaciones

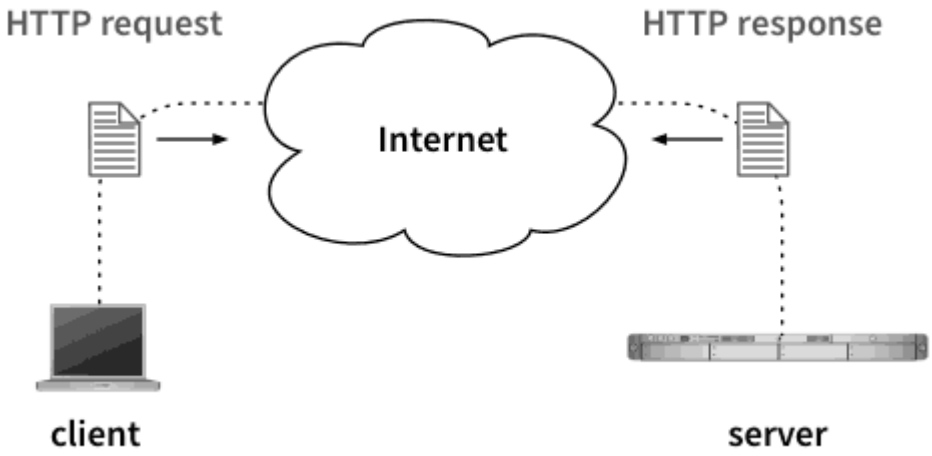
El Protocolo de transferencia de hipertexto (HTTP) utiliza un modelo de solicitud de cliente / respuesta de servidor. HTTP es un protocolo sin estado, lo que significa que no requiere que el servidor retenga información o estado sobre cada usuario durante la duración de múltiples solicitudes. Sin embargo, por razones de rendimiento y para evitar problemas de latencia de conexión de TCP, se pueden usar técnicas como las conexiones persistentes, paralelas o segmentadas.

Versiones

Versión	Nota (s)	Fecha estimada de lanzamiento
HTTP / 0.9	"Según lo implementado"	1991-01-01
HTTP / 1.0	Primera versión de HTTP / 1.0, última versión que no se ha convertido en un RFC	1992-01-01
HTTP / 1.0 r1	Primer RFC oficial para HTTP	1996-05-01
HTTP / 1.1	Mejoras en el manejo de la conexión, soporte para hosts virtuales basados en nombres	1997-01-01
HTTP / 1.1 r1	Se eliminó el uso de palabras clave no ambiguas, se solucionaron posibles problemas con el enmarcado de mensajes	1999-06-01
HTTP / 1.1 r2	Grandes reparaciones	2014-06-01
HTTP / 2	Primera especificación para HTTP / 2	2015-05-01

Examples

Solicitudes y respuestas HTTP



HTTP describe cómo un cliente HTTP, como un navegador web, envía una solicitud HTTP a través de una red a un servidor HTTP, que luego envía una respuesta HTTP al cliente.

La solicitud HTTP suele ser una solicitud de un recurso en línea, como una página web o una imagen, pero también puede incluir información adicional, como los datos ingresados en un formulario. La respuesta HTTP suele ser una representación de un recurso en línea, como una página web o una imagen.

HTTP / 1.0

HTTP / 1.0 fue descrito en RFC 1945 .

HTTP / 1.0 no tiene algunas características que hoy en día se requieren de facto en la Web, como el encabezado `Host` para hosts virtuales.

Sin embargo, los clientes y servidores HTTP a veces aún declaran que usan HTTP / 1.0 si tienen una implementación incompleta del protocolo HTTP / 1.1 (por ejemplo, sin codificación de transferencia fragmentada o segmentación), o la compatibilidad se considera más importante que el rendimiento (por ejemplo, cuando se conecta a un proxy local). servidores).

```
GET / HTTP/1.0
User-Agent: example/1

HTTP/1.0 200 OK
Content-Type: text/plain

Hello
```

HTTP / 1.1

HTTP / 1.1 se especificó originalmente en 1999 en RFC 2616 (protocolo) y RFC 2617 (autenticación), pero estos documentos ahora están obsoletos y no deben usarse como referencia:

> No utilice RFC2616. Elimínelo de sus discos duros, marcadores y grabe (o recicle responsablemente) cualquier copia que se imprima.

- Mark Nottingham, presidente del HTTP WG

La especificación actualizada de HTTP / 1.1, que coincide con la forma en que se implementa HTTP en la actualidad, se encuentra en los nuevos RFC 723x:

- RFC 7230: Sintaxis de mensajes y enrutamiento
- RFC 7231: Semántica y Contenido
- RFC 7232: Solicitudes Condicionales
- RFC 7233: Solicitudes de rango
- RFC 7234: almacenamiento en caché
- RFC 7235: Autenticación

HTTP / 1.1 agregado, entre otras características:

- codificación de transferencia fragmentada, que permite a los servidores enviar de manera confiable respuestas de tamaño desconocido,
- conexiones TCP / IP persistentes (que no eran una extensión estándar en HTTP / 1.0),
- rango de solicitudes utilizadas para reanudar descargas,
- control de caché.

HTTP / 1.1 intentó introducir la canalización, lo que permitía a los clientes HTTP reducir la latencia de solicitud-respuesta al enviar varias solicitudes a la vez sin esperar respuestas. Desafortunadamente, esta función nunca se implementó correctamente en algunos proxies, lo que provocó que las conexiones canalizadas dejaran o reordenaran las respuestas.

```
GET / HTTP/1.0
User-Agent: example/1
Host: example.com

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 6
Connection: close

Hello
```

HTTP / 2

HTTP / 2 (RFC 7540) cambió el formato HTTP de una simple solicitud basada en texto y los encabezados de respuesta al formato de datos binarios enviados en marcos. HTTP / 2 soporta la compresión de los encabezados (HPACK).

Esto redujo la sobrecarga de solicitudes y permitió recibir múltiples respuestas simultáneamente en una sola conexión TCP / IP.

A pesar de los grandes cambios en el formato de los datos, HTTP / 2 todavía usa encabezados HTTP, y las solicitudes y respuestas se pueden traducir con precisión entre HTTP / 1.1 y 2.

HTTP / 0.9

La primera versión de HTTP que entró en existencia es 0.9, a menudo llamada " HTTP como implementado ". Una descripción común de 0.9 es "una subsección del protocolo HTTP [es decir, 1.0] completo". Sin embargo, esto no ilustra en gran medida la disparidad en las capacidades entre 0.9 y 1.0.

Ni peticiones ni respuestas en 0.9 encabezados de función. Las solicitudes consisten en una sola línea de `GET` terminada en CRLF, seguida de un espacio, seguida de la URL del recurso solicitado. Se espera que las respuestas sean un solo documento HTML. El final de dicho documento se marca al quitar la conexión del lado del servidor. No hay instalaciones para indicar el éxito o el fracaso de una operación. La única propiedad interactiva es la cadena de búsqueda que está estrechamente vinculada a la etiqueta HTML `<isindex>` .

El uso de HTTP / 0.9 es hoy en día excepcionalmente raro. Ocasionalmente se ve en sistemas embebidos como una alternativa a tftp .

Lea Empezando con HTTP en línea: https://riptutorial.com/es/http/topic/984/empezando-con-http

Capítulo 2: Autenticación

Parámetros

Parámetro	Detalles
Estado de respuesta	`401` si el servidor de origen requiere autenticación, `407` si un proxy intermedio requiere autenticación
Cabeceras de respuesta	`WWW-Authenticate` por el servidor de origen, `Proxy-Authenticate` por un proxy intermedio
Solicitar encabezados	`Authorization` de autorización contra un servidor de origen, `Proxy-Authorization` contra un proxy intermedio
Esquema de autentificación	`Basic` para la autenticación básica, pero se pueden usar otros como `Digest` y `SPNEGO` . Ver el registro de esquemas de autenticación HTTP .
Reino	Un nombre del espacio protegido en el servidor; un servidor puede tener múltiples espacios de este tipo, cada uno con un nombre distinto y mecanismos de autenticación.
Cartas credenciales	Para `Basic` : nombre de usuario y contraseña separados por dos puntos, codificados en base64; por ejemplo, `username:password` codificada en base64 es `dXNlcm5hbWU6cGFzc3dvcmQ=`

Observaciones

La autenticación básica se define en RFC2617 . Se puede usar para autenticarse en el servidor de origen después de recibir un `401 Unauthorized` , así como en un servidor proxy después de un `407 (Proxy Authentication Required)` . En las credenciales (decodificadas), la contraseña comienza después de los primeros dos puntos. Por lo tanto, el nombre de usuario no puede contener dos puntos, pero la contraseña puede.

Examples

Autenticación básica HTTP

La autenticación básica HTTP proporciona un mecanismo sencillo para la autenticación. Las credenciales se envían en texto sin formato y, por lo tanto, son inseguras de forma predeterminada. La autenticación exitosa procede de la siguiente manera.

El cliente solicita una página para la cual el acceso está restringido:

```
GET /secret
```

El servidor responde con el código de estado `401 Unauthorized` y solicita al cliente que se autentique:

```
401 Unauthorized
WWW-Authenticate: Basic realm="Secret Page"
```

El cliente envía el encabezado de `Authorization` . Las credenciales son `username:password` codificada en base64:

```
GET /secret
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

El servidor acepta las credenciales y responde con el contenido de la página:

```
HTTP/1.1 200 OK
```

Lea Autenticación en línea: https://riptutorial.com/es/http/topic/3286/autenticacion

Capítulo 3: Cachear las respuestas HTTP

Observaciones

Las respuestas se almacenan en caché por separado para cada URL y cada método HTTP.

El almacenamiento en caché de HTTP se define en RFC 7234 .

Glosario

- **fresco** : estado de una respuesta en caché, que aún no ha caducado. Normalmente, una respuesta nueva puede *satisfacer las* solicitudes sin necesidad de contactar con el servidor del que se originó la respuesta.
- **obsoleto** : estado de una respuesta almacenada en caché, que ha pasado su fecha de caducidad. Normalmente, las respuestas obsoletas no pueden usarse para *satisfacer* una solicitud sin verificar con el servidor si sigue siendo válida.
- **satisfacer** : la respuesta en caché *satisface* una solicitud cuando todas las condiciones en la solicitud coinciden con la respuesta en caché, por ejemplo, tienen el mismo método HTTP y URL, la respuesta es nueva o la solicitud permite respuestas obsoletas, los encabezados de solicitud coinciden con los encabezados enumerados en el encabezado `Vary` la respuesta, etc. .
- **revalidación** : comprobar si una respuesta en caché es nueva. Esto generalmente se hace con una *solicitud condicional que* contiene `If-Modified-Since` o `If-None-Match` y el estado de respuesta `304` .
- **caché privado** : caché para un solo usuario, por ejemplo, en un navegador web. Los cachés privados pueden almacenar respuestas personalizadas.
- **caché pública** : caché compartida entre muchos usuarios, por ejemplo, en un servidor proxy. Tal caché puede enviar la misma respuesta a múltiples usuarios.

Examples

Respuesta de caché para todos por 1 año.

```
Cache-Control: public, max-age=31536000
```

`public` significa que la respuesta es la misma para todos los usuarios (no contiene ninguna información personalizada). `max-age` es en segundos a partir de ahora. 31536000 = 60 * 60 * 24 * 365.

Esto se recomienda para activos estáticos que nunca deben cambiar.

Caché de respuesta personalizada por 1 minuto.

```
Cache-Control: private, max-age=60
```

`private` especifica que la respuesta se puede almacenar en caché solo para el usuario que solicitó el recurso y no se puede reutilizar cuando otros usuarios solicitan el mismo recurso. Esto es apropiado para respuestas que dependen de las cookies.

Deje de usar los recursos en caché sin consultar primero con el servidor

```
Cache-Control: no-cache
```

El cliente se comportará como si la respuesta no estuviera en caché. Esto es apropiado para recursos que pueden cambiar de forma impredecible en cualquier momento, y que los usuarios siempre deben ver en la última versión.

Las respuestas con `no-cache` serán más lentas (latencia alta) debido a la necesidad de contactar al servidor cada vez que se usan.

Sin embargo, para ahorrar ancho de banda, los clientes aún *pueden* almacenar dichas respuestas. Las respuestas con `no-cache` no se utilizarán para satisfacer las solicitudes sin ponerse en contacto con el servidor cada vez que se compruebe si la respuesta en caché se puede reutilizar.

Solicitar respuestas para no ser almacenadas en absoluto

```
Cache-control: no-store
```

Indica a los clientes que no almacenen en caché la respuesta de ninguna manera y que la olviden lo antes posible.

Esta directiva se diseñó originalmente para datos confidenciales (hoy se debe usar HTTPS en su lugar), pero se puede usar para evitar caches contaminantes con respuestas que no se pueden reutilizar.

Es apropiado solo en casos específicos donde los datos de respuesta siempre son diferentes, por ejemplo, un punto final de API que devuelve un gran número aleatorio. De lo contrario, `no-cache` se puede usar la `no-cache` y la revalidación para tener un comportamiento de respuesta "no almacenable", al mismo tiempo que se puede ahorrar algo de ancho de banda.

Cabeceras obsoletas, redundantes y no estándar

- `Expires` : especifica la fecha en que el recurso se vuelve obsoleto. Se basa en servidores y clientes que tienen relojes precisos y zonas horarias compatibles correctamente. `Cache-control: max-age` tiene prioridad sobre `Expires` , y generalmente es más confiable.
- `post-check` directivas `post-check` y `pre-check` son extensiones no estándar de Internet Explorer que permiten el uso de respuestas obsoletas. La alternativa estándar es `stale-while-revalidate` .
- `Pragma: no-cache` - obsoleto en 1999 . `Cache-control` debe utilizarse en su lugar.

Cambio de recursos en caché

El método más sencillo para omitir el caché es cambiar la URL. Esto se usa como una mejor práctica cuando la URL contiene una versión o una suma de comprobación del recurso, por ejemplo

```
http://example.com/image.png?version=1
http://example.com/image.png?version=2
```

Estas dos URL se almacenarán en caché por separado, por lo que incluso si `…?version=1` se almacenó en caché *para siempre* , una nueva copia podría recuperarse inmediatamente como `…?version=2` .

Por favor, no utilice URLs aleatorias para evitar cachés. Use `Cache-control: no-cache` o `Cache-control: no-store` lugar. Si las respuestas con direcciones URL aleatorias se envían sin la directiva de `no-store` , se almacenarán innecesariamente en cachés y expulsarán respuestas más útiles fuera de la memoria caché, lo que degradará el rendimiento de toda la memoria caché.

Lea Cachear las respuestas HTTP en línea: https://riptutorial.com/es/http/topic/3296/cachear-las-respuestas-http

Capítulo 4: Codificaciones de respuesta y compresión.

Examples

Compresión HTTP

El cuerpo del mensaje HTTP se puede comprimir (desde HTTP / 1.1). El servidor comprime la solicitud y agrega un encabezado de `Content-Encoding` , o lo hace un proxy y agrega un encabezado de `Transfer-Encoding` .

Un cliente puede enviar un encabezado de solicitud de `Accept-Encoding` para indicar qué codificaciones acepta.

Las codificaciones más utilizadas son:

- gzip: algoritmo de desinflado (LZ77) con suma de comprobación CRC32 implementado en el programa de compresión del archivo "gzip" (RFC1952)
- desinflar - formato de datos "zlib" (RFC1950), algoritmo de desinflado (híbrido LZ77 y Huffman) con suma de comprobación Adler32

Múltiples métodos de compresión.

Es posible comprimir el cuerpo de un mensaje de respuesta HTTP más de una vez. Los nombres de codificación deben estar separados por una coma en el orden en que se aplicaron. Por ejemplo, si un mensaje se ha comprimido a través de deflate y luego gzip, el encabezado debería verse así:

```
Content-Encoding: deflate, gzip
```

Varios encabezados de `Content-Encoding` también son válidos, aunque no se recomiendan:

```
Content-Encoding: deflate
Content-Encoding: gzip
```

compresión gzip

El cliente primero envía una solicitud con un encabezado `Accept-Encoding` que indica que admite gzip:

```
GET / HTTP/1.1\r\n
Host: www.google.com\r\n
Accept-Encoding: gzip, deflate\r\n
\r\n
```

El servidor puede enviar una respuesta con un cuerpo de respuesta comprimido y un encabezado de `Content-Encoding` que especifica que se usó la codificación gzip:

```
HTTP/1.1 200 OK\r\n
Content-Encoding: gzip\r\n
Content-Length: XX\r\n
\r\n
... compressed content ...
```

Lea Codificaciones de respuesta y compresión. en línea: https://riptutorial.com/es/http/topic/5046/codificaciones-de-respuesta-y-compresion-

Capítulo 5: Códigos de estado HTTP

Introducción

En HTTP, los códigos de estado son un mecanismo legible por máquina que indica el resultado de una solicitud emitida anteriormente. De RFC 7231, sec. 6 : "El elemento de código de estado es un código entero de tres dígitos que da el resultado del intento de comprender y satisfacer la solicitud".

La gramática formal permite que los códigos estén entre `000` y `999` . Sin embargo, solo el rango de `100` a `599` tiene un significado asignado.

Observaciones

HTTP / 1.1 define una serie de códigos de estado HTTP numéricos que aparecen en la línea de estado (la primera línea de una respuesta HTTP) para resumir lo que el cliente debe hacer con la respuesta.

El primer dígito de los códigos de estado define la clase de la respuesta:

- `1xx` informativo
- Solicitud de cliente `2xx` exitosa
- Solicitud `3xx` redirigida : se requieren acciones adicionales, como una nueva solicitud
- `4xx` Error de cliente - no repetir la misma solicitud
- `5xx` error del servidor - tal vez intente de nuevo

En la práctica, no siempre es fácil elegir el código de estado más apropiado.

Examples

Error interno de servidor 500

Un **error interno del servidor HTTP 500** es un mensaje general que significa que el servidor encontró algo inesperado. Las aplicaciones (o el servidor web general) deben usar un 500 cuando se produce un error al procesar la solicitud, es decir, se produce una excepción o una condición del recurso impide que se complete el proceso.

Ejemplo de línea de estado:

```
HTTP/1.1 500 Internal Server Error
```

404 No encontrado

HTTP 404 No encontrado significa que el servidor no pudo encontrar la ruta utilizando el URI que el cliente solicitó.

```
HTTP/1.1 404 Not Found
```

La mayoría de las veces, el archivo solicitado se eliminó, pero a veces puede ser una mala configuración de la raíz del documento o una falta de permisos (aunque los permisos faltantes activan HTTP 403 Forbidden con mayor frecuencia).

Por ejemplo, el IIS de Microsoft escribe 404.0 (0 es el subestado) en sus archivos de registro cuando se eliminó el archivo solicitado. Pero cuando la solicitud entrante está bloqueada por las reglas de filtrado de solicitudes, escribe 404.5-404.19 para registrar los archivos de acuerdo con la regla que bloquea la solicitud. Puede encontrar una referencia al código de error más detallada en el Soporte de Microsoft .

Denegación de acceso a archivos protegidos

Use 403 Prohibido cuando un cliente ha solicitado un recurso al que no se puede acceder debido a los controles de acceso existentes. Por ejemplo, si su aplicación tiene una ruta `/admin` que solo debería ser accesible para usuarios con derechos administrativos, puede usar 403 cuando un usuario normal solicita la página.

```
GET /admin HTTP/1.1
Host: example.com
```

```
HTTP/1.1 403 Forbidden
```

Solicitud exitosa

Envíe una respuesta HTTP con el código de estado `200` para indicar una solicitud exitosa. La línea de estado de respuesta HTTP es entonces:

```
HTTP/1.1 200 OK
```

El texto de estado `OK` es solo informativo. El cuerpo de la respuesta (carga útil del mensaje) debe contener una representación del recurso solicitado. Si no hay representación 201 No se debe usar Contenido.

Respondiendo a una solicitud condicional de contenido en caché

Enviar un estado de respuesta **304 No modificado** desde el servidor enviar en respuesta a una solicitud de cliente que contiene encabezados `If-Modified-Since` y `If-None-Match` , si el recurso de solicitud no ha cambiado.

Por ejemplo, si una solicitud de un cliente para una página web incluye el encabezado `If-Modified-Since: Fri, 22 Jul 2016 14:34:40 GMT` y la página no se modificó desde entonces, responda con la línea de estado `HTTP/1.1 304 Not Modified`

Top 10 código de estado HTTP

2xx exito

- **200 OK** - Respuesta estándar para solicitudes HTTP exitosas.
- **201 Creado** : la solicitud se ha cumplido, dando como resultado la creación de un nuevo recurso.
- **204 Sin contenido** : el servidor procesó correctamente la solicitud y no devuelve ningún contenido.

Redireccionamiento 3xx

- **304 No modificado** : indica que el recurso no se ha modificado desde la versión especificada por los encabezados de solicitud `If-Modified-Since` o `If-None-Match` .

Error del cliente 4xx

- **400 Solicitud errónea** : el servidor no puede o no procesará la solicitud debido a un error aparente del cliente (por ejemplo, sintaxis de solicitud mal formada, tamaño demasiado grande, tramado de mensajes de solicitud no válida o enrutamiento de solicitud engañosa).
- **401 no autorizado** : *similar a 403 prohibido* , pero específicamente para uso cuando se requiere autenticación y ha fallado o aún no se ha proporcionado. La respuesta debe incluir un campo de encabezado `WWW-Authenticate` contenga un desafío aplicable al recurso solicitado.
- **403 Prohibido** : la solicitud era válida, pero el servidor se niega a responder. Es posible que el usuario haya iniciado sesión pero no tenga los permisos necesarios para el recurso.
- **404 No encontrado** : el recurso solicitado no se pudo encontrar pero podría estar disponible en el futuro. Se permiten solicitudes posteriores por parte del cliente.
- **409 Conflicto** : indica que la solicitud no se pudo procesar debido a un conflicto en la solicitud, como un conflicto de edición entre varias actualizaciones simultáneas.

Error del servidor 5xx

- **500 Error interno del servidor** : un mensaje de error genérico, dado cuando se encontró una condición inesperada y no es adecuado ningún mensaje más específico.

Lea Códigos de estado HTTP en línea: https://riptutorial.com/es/http/topic/2577/codigos-de-estado-http

Capítulo 6: HTTP para APIs

Observaciones

Las API HTTP utilizan un amplio espectro de verbos HTTP y, por lo general, devuelven respuestas JSON o XML.

Examples

Crear un recurso

No todos están de acuerdo en cuál es el método más semánticamente correcto para la creación de recursos. Por lo tanto, su API podría aceptar `POST` o `PUT` , o cualquiera.

El servidor debe responder con `201 Created` si el recurso se creó correctamente. Elija el código de error más apropiado si no lo era.

Por ejemplo, si proporciona una API para crear registros de empleados, la solicitud / respuesta podría tener este aspecto:

```
POST /employees HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Charlie Smith",
    "age": 38,
    "job_title": "Software Developer",
    "salary": 54895.00
}
```

```
HTTP/1.1 201 Created
Location: /employees/1/charlie-smith
Content-Type: application/json

{
    "employee": {
        "name": "Charlie Smith",
        "age": 38,
        "job_title": "Software Developer",
        "salary": 54895.00
        "links": [
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "self",
                "method": "GET"
            },
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "delete",
                "method": "DELETE"
            },
```

```
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "edit",
                "method": "PATCH"
            }
        ]
    },
    "links": [
        {
            "uri": "/employees",
            "rel": "create",
            "method": "POST"
        }
    ]
}
```

La inclusión de los campos JSON de `links` en la respuesta permite al cliente acceder al recurso relacionado con el nuevo recurso y con la aplicación en su conjunto, sin tener que conocer sus URI o métodos de antemano.

Editar un recurso

Editar o actualizar un recurso es un propósito común de las API. Las ediciones se pueden lograr mediante el envío de `POST` , `PUT` o `PATCH` al recurso respectivo. Aunque `POST` tiene permitido agregar datos a la representación existente de un recurso, se recomienda usar `PUT` o `PATCH` ya que transmiten una semántica más explícita.

Su servidor debe responder con `200 OK` si se realizó la actualización, o `202 Accepted` si aún no se ha aplicado. Elija el código de error más apropiado si no se puede completar.

Actualizaciones completas

`PUT` tiene la semántica de reemplazar la representación actual con la carga útil incluida en la solicitud. Si la carga útil no es del mismo tipo de representación que la representación actual del recurso a actualizar, el servidor puede decidir qué enfoque tomar. RFC7231 define que el servidor puede:

- Reconfigure el recurso de destino para reflejar el nuevo tipo de medio
- Transforme la representación PUT a un formato consistente con el del recurso antes de guardarlo como el nuevo estado de recurso
- Rechace la solicitud con una respuesta de `415 Unsupported Media Type` que indique que el recurso de destino está limitado a un determinado (conjunto) de tipos de medios.

Un recurso base que contiene una representación JSON HAL como ...

```
{
    "name": "Charlie Smith",
    "age": 39,
    "job_title": "Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
```

```
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    }
}
```

... puede recibir una solicitud de actualización como esta

```
PUT /users/1234 HTTP/1.1
Host: http://www.acmee.com
Content-Type: "application/json; charset=utf-8"
Content-Length: 85
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer"
}
```

El servidor ahora puede reemplazar el estado del recurso con el cuerpo de solicitud dado y también cambiar el tipo de contenido de `application/hal+json` a `application/json` o convertir la carga útil JSON en una representación JSON HAL y luego reemplazar el contenido del recurso con la transformada o rechace la solicitud de actualización debido a un tipo de medio inaplicable con una respuesta de `415 Unsupported Media Type` .

Hay una diferencia entre reemplazar el contenido directamente o primero transformar la representación en el modelo de representación definido y luego reemplazar el contenido existente con el transformado. Una solicitud `GET` posterior devolverá la siguiente respuesta en un reemplazo directo:

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer"
}
```

mientras que el enfoque de transformación y luego reemplazo devolverá la siguiente representación:

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept-Encoding: gzip, deflate
```

```
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: e0023aa4e

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    }
}
```

Efectos secundarios

Tenga en cuenta que `PUT` puede tener efectos secundarios, aunque se define como una operación idempotente. Esto está documentado en RFC7231 como

> Una solicitud PUT aplicada al recurso de destino **puede tener efectos secundarios en otros recursos** . Por ejemplo, un artículo puede tener un URI para identificar "la versión actual" (un recurso) que es independiente de los URI que identifican cada versión en particular (diferentes recursos que en un punto compartían el mismo estado que el recurso de la versión actual). Por lo tanto, una solicitud PUT exitosa en el URI de "la versión actual" podría crear un nuevo recurso de versión además de cambiar el estado del recurso de destino, y también podría ocasionar que se agreguen enlaces entre los recursos relacionados.

La producción de entradas de registro adicionales no se considera un efecto secundario, ya que este no es un estado de un recurso en general.

Actualizaciones parciales

RFC7231 menciona esto con respecto a actualizaciones parciales:

> Las actualizaciones parciales de contenido son posibles al apuntar a un recurso identificado por separado con un estado que se superpone a una parte del recurso más grande, o al usar un método diferente que se haya definido específicamente para actualizaciones parciales (por ejemplo, el método PATCH definido en RFC5789).

Por lo tanto, las actualizaciones parciales se pueden realizar en dos sabores:

- Haga que un recurso incruste múltiples sub-recursos más pequeños y actualice solo un sub-recurso respectivo en lugar del recurso completo a través de `PUT`
- Usando `PATCH` e instruir al servidor que actualizar

Actualización parcial con estado superpuesto

Si una representación de usuario necesita actualizarse parcialmente debido a un traslado de un usuario a otra ubicación, en lugar de actualizar al usuario directamente, el recurso relacionado debe actualizarse directamente, lo que se refleja en una actualización parcial de la representación del usuario.

Antes de la mudanza un usuario tenía la siguiente representación.

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept: application/hal+json; charset=utf-8
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    },
    "_embedded": {
        "ea:address": {
            "street": "Terrace Drive, Central Park",
            "zip": "NY 10024"
            "city": "New York",
            "country": "United States of America",
            "_links": {
                "self": { "href": "/address/abc" },
                "google_maps": { "href": "http://maps.google.com/?ll=40.7739166,-73.970176" }
            }
        }
    }
}
```

Cuando el usuario se está moviendo a una nueva ubicación, actualiza la información de su ubicación de esta manera:

```
PUT /address/abc HTTP/1.1
Host: http://www.acmee.com
Content-Type: "application/json; charset=utf-8"
Content-Length: 109
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

{
    "street": "Standford Ave",
```

```
    "zip": "CA 94306",
    "city": "Pablo Alto",
    "country": "United States of America"
}
```

Con la transformación antes de reemplazar la semántica para el tipo de medio no coincidente entre el recurso de dirección existente y el que está en la solicitud, como se describió anteriormente, el recurso de dirección ahora se actualiza, lo que tiene el efecto que en una solicitud `GET` posterior en el recurso de usuario Se devuelve la nueva dirección para el usuario.

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept: application/hal+json; charset=utf-8
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    },
    "_embedded": {
        "ea:address": {
            "street": "Standford Ave",
            "zip": "CA 94306",
            "city": "Pablo Alto",
            "country": "United States of America"
            "_links": {
                "self": { "href": "/address/abc" },
                "google_maps": { "href": "http://maps.google.com/?ll=37.4241311,-122.1524475"
}
            }
        }
    }
}
```

Parcheando datos parciales

`PATCH` se define en RFC5789 y no forma parte directamente de la especificación de HTTP per se. Un error común es que enviar solo los campos que deben actualizarse parcialmente es suficiente dentro de una solicitud de `PATCH` . Por lo tanto, la especificación establece

> El método PATCH solicita que un conjunto de cambios descritos en la entidad de solicitud se aplique al recurso identificado por el URI de solicitud. El conjunto de

cambios se representa en un formato denominado "documento de revisión" identificado por un tipo de medio.

Esto significa que un cliente debe calcular los pasos necesarios necesarios para transformar el recurso del estado A al estado B y enviar estas instrucciones al servidor.

Un tipo de medio popular basado en JSON para la aplicación de parches es JSON Patch .

Si la edad y el título del trabajo de nuestro usuario de muestra cambian y se debe agregar un campo adicional que represente los ingresos del usuario, se debe agregar una actualización parcial utilizando `PATCH` utilizando el parche JSON:

```
PATCH /users/1234 HTTP/1.1
Host: http://www.acmee.com
Content-Type: application/json-patch+json; charset=utf-8
Content-Length: 188
Accept: application/json
If-Match: "e0023aa4e"

[
    { "op": "replace", "path": "/age", "value": 40 },
    { "op": "replace", "path": "/job_title", "value": "Senior Software Developer" },
    { "op": "add", "path": "/salery", "value": 63985.00 }
]
```

`PATCH` puede actualizar múltiples recursos a la vez y requiere aplicar los cambios de forma atómica, lo que significa que se deben aplicar todos los cambios o ninguno en absoluto, lo que supone una carga transaccional para el implementador de la API.

Una actualización exitosa puede devolver algo como esto

```
HTTP/1.1 200 OK
Location: /users/1234
Content-Type: application/json
ETag: "df00eb258"

{
    "name": "Charlie Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "salary": 63985.00
}
```

aunque no está restringido a `200 OK` códigos de respuesta `200 OK` solamente.

Para evitar las actualizaciones intermedias (los cambios realizados entre la recuperación previa del estado de representación y la actualización) se deben utilizar los encabezados `ETag` , `If-Match` o `If-Unmodified-Since` .

Manejo de errores

La especificación en `PATCH` recomienda el siguiente manejo de errores:

Tipo	Código de error
Documento de parche mal formado	`400 Bad Request`
Documento de parche no admitido	`415 Unsupported Media Type`
Solicitud no procesable, es decir, si el recurso se vuelve inválido al aplicar el parche	`422 Unprocessable Entity`
Recurso no encontrado	`404 Not Found`
Estado conflictivo, es decir, un cambio de nombre (movimiento) de un campo que no existe	`409 Conflict`
Modificación conflictiva, es decir, si el cliente utiliza un encabezado `If-Match` o `If-Unmodified-Since` cuya validación falló. Si no se dispone de una condición previa, se debe devolver el último código de error	`412 Precondition Failed` o `409 Conflict`
Modificación concurrente, es decir, si la solicitud debe aplicarse antes de la aceptación, más solicitudes de `PATCH`	`409 Conflict`

Eliminar un recurso

Otro uso común de las API de HTTP es eliminar un recurso existente. Esto usualmente se debe hacer usando las solicitudes `DELETE` .

Si la eliminación fue exitosa, el servidor debería devolver `200 OK` ; un código de error apropiado si no lo era.

Si nuestro empleado Charlie Smith ha dejado la empresa y ahora queremos eliminar sus registros, podría tener este aspecto:

```
DELETE /employees/1/charlie-smith HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    'links': [
        {
            'uri': '/employees',
            'rel': 'create',
            'method': 'POST'
        }
    ]
}
```

Lista de recursos

El último uso común de las API de HTTP es obtener una lista de los recursos existentes en el

servidor. Las listas de este tipo deben obtenerse utilizando solicitudes `GET` , ya que solo *recuperan* datos.

El servidor debe devolver `200 OK` si puede suministrar la lista, o un código de error apropiado si no lo hace.

Listado de nuestros empleados, entonces, podría verse así:

```
GET /employees HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    'employees': [
        {
            'name': 'Charlie Smith',
            'age': 39,
            'job_title': 'Software Developer',
            'salary': 63985.00
            'links': [
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'self',
                    'method': 'GET'
                },
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'delete',
                    'method': 'DELETE'
                },
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'edit',
                    'method': 'PATCH'
                }
            ]
        },
        {
            'name': 'Donna Prima',
            'age': 30,
            'job_title': 'QA Tester',
            'salary': 77095.00
            'links': [
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'self',
                    'method': 'GET'
                },
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'delete',
                    'method': 'DELETE'
                },
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'edit',
                    'method': 'PATCH'
```

```
                }
            ]
        }
    ],
    'links': [
        {
            'uri': '/employees/new',
            'rel': 'create',
            'method': 'PUT'
        }
    ]
}
```

Lea HTTP para APIs en línea: https://riptutorial.com/es/http/topic/3423/http-para-apis

Capítulo 7: Origen cruzado y control de acceso

Observaciones

El intercambio de recursos de origen cruzado está diseñado para permitir solicitudes dinámicas entre dominios, a menudo utilizando técnicas como AJAX . Mientras que las secuencias de comandos realizan la mayor parte del trabajo, el servidor HTTP debe admitir la solicitud utilizando los encabezados correctos.

Examples

Cliente: enviar una solicitud de intercambio de recursos de origen cruzado (CORS)

Se debe enviar una *solicitud de* origen cruzado que incluya el encabezado de `Origin` . Esto indica de donde se originó la solicitud. Por ejemplo, una solicitud de origen cruzado de `http://example.com` a `http://example.org` se vería así:

```
GET /cors HTTP/1.1
Host: example.org
Origin: example.com
```

El servidor utilizará este valor para determinar si la solicitud está autorizada.

Servidor: respondiendo a una solicitud CORS

La respuesta a una solicitud CORS debe incluir un encabezado `Access-Control-Allow-Origin` , que dicta qué orígenes pueden usar el recurso CORS. Este encabezado puede tomar uno de tres valores:

- Un origen. Hacer esto solo permite solicitudes desde *ese origen* .
- El personaje `*` . Esto permite solicitudes desde *cualquier origen* .
- La cadena `null` . Esto *no* permite *solicitudes de CORS* .

Por ejemplo, al recibir una solicitud CORS del origen `http://example.com` , si `example.com` es un origen autorizado, el servidor enviaría esta respuesta:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.com
```

Una respuesta de cualquier origen también permitiría esta solicitud, es decir:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: *
```

Permitir credenciales de usuario o sesión

Al permitir que las credenciales del usuario o la sesión del usuario se envíen con una solicitud CORS, el servidor puede conservar los datos del usuario en todas las solicitudes CORS. Esto es útil si el servidor necesita verificar si el usuario ha iniciado sesión antes de proporcionar datos (por ejemplo, solo realiza una acción si un usuario ha iniciado sesión; esto requeriría que la solicitud CORS se envíe con credenciales).

Esto se puede lograr en el lado del servidor para solicitudes con verificación previa, mediante el envío del encabezado `Access-Control-Allow-Credentials` en respuesta a la solicitud de verificación previa de `OPTIONS` . Tome el siguiente caso de una solicitud CORS para `DELETE` un recurso:

```
OPTIONS /cors HTTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: DELETE
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.org
Access-Control-Allow-Methods: DELETE
Access-Control-Allow-Credentials: true
```

La línea `Access-Control-Allow-Credentials: true` indica que la siguiente solicitud `DELETE` CORS puede enviarse con las credenciales del usuario.

Solicitudes de verificación previa

Se permite una solicitud CORS básica para usar uno de solo dos métodos:

- OBTENER
- ENVIAR

y solo unos pocos encabezados selectos. Las solicitudes POST CORS pueden elegir adicionalmente solo entre tres tipos de contenido.

Para evitar este problema, las solicitudes que deseen utilizar otros métodos, encabezados o tipos de contenido deben emitir primero una solicitud de *verificación* previa, que es una solicitud de `OPTIONS` que incluye encabezados de solicitud de control de acceso. Por ejemplo, esta es una solicitud de verificación previa que verifica si el servidor aceptará una solicitud `PUT` que incluya un encabezado `DNT` :

```
OPTIONS /cors HTTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: DNT
```

Servidor: respondiendo a las solicitudes de verificación previa

Cuando un servidor recibe una solicitud de verificación previa, debe verificar si es compatible con el método y los encabezados solicitados, y devolver una respuesta que indique su capacidad para respaldar la solicitud, así como cualquier otro dato permitido (como credenciales).

Estos están indicados en el control de acceso Permitir encabezados. El servidor también puede devolver un encabezado de `Max-Age` control de acceso, que indica cuánto tiempo se puede almacenar en caché la respuesta de verificación previa.

Esto es lo que podría parecer un ciclo de solicitud-respuesta para una solicitud de verificación previa:

```
OPTIONS /cors HHTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: DNT
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.org
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: DNT
```

Lea Origen cruzado y control de acceso en línea: https://riptutorial.com/es/http/topic/3424/origen-cruzado-y-control-de-acceso

Capítulo 8: Peticiones HTTP

Parámetros

Método HTTP	Propósito
`OPTIONS`	Recupere información sobre las opciones de comunicación (métodos disponibles y encabezados) disponibles en el URI de solicitud especificado.
`GET`	Recupere los datos identificados por el URI de solicitud o los datos producidos por el script disponible en el URI de solicitud.
`HEAD`	Idéntico a `GET` excepto que el servidor no devolverá ningún cuerpo de mensaje: solo encabezados.
`POST`	Envíe un bloque de datos (especificado en el cuerpo del mensaje) al servidor para agregarlos al recurso especificado en el URI de la solicitud. Más comúnmente utilizado para el procesamiento de formularios.
`PUT`	Almacene la información adjunta (en el cuerpo del mensaje) como un recurso nuevo o actualizado bajo el URI de solicitud dado.
`DELETE`	Eliminar, o poner en cola para la eliminación, el recurso identificado por el URI de la solicitud.
`TRACE`	Esencialmente, un comando echo: un servidor HTTP compatible y en funcionamiento debe enviar la solicitud completa de vuelta como el cuerpo de una respuesta 200 (OK).

Observaciones

El método `CONNECT` está reservado por la especificación de las definiciones de métodos para su uso con proxies que pueden cambiar entre los modos de proxy y túnel (por ejemplo, para el túnel SSL).

Examples

Enviar una solicitud HTTP mínima manualmente utilizando Telnet

Este ejemplo demuestra que HTTP es un protocolo de comunicaciones de Internet basado en texto y muestra una solicitud HTTP básica y la respuesta HTTP correspondiente.

Puede usar Telnet para enviar manualmente una solicitud HTTP mínima desde la línea de comandos, de la siguiente manera.

1. Inicie una sesión de Telnet en el servidor web `www.example.org` en el puerto 80:

   ```
   telnet www.example.org 80
   ```

   Telnet informa que se ha conectado al servidor:

   ```
   Connected to www.example.org.
   Escape character is '^]'.
   ```

2. Ingrese una línea de solicitud para enviar una ruta URL de solicitud GET `/` , usando HTTP 1.1

   ```
   GET / HTTP/1.1
   ```

3. Ingrese una línea de campo de encabezado HTTP para identificar la parte del nombre de host de la URL requerida, que se requiere en HTTP 1.1

   ```
   Host: www.example.org
   ```

4. Ingrese una línea en blanco para completar la solicitud.

   El servidor web envía la respuesta HTTP, que aparece en la sesión de Telnet.

La sesión completa, es la siguiente. La primera línea de la respuesta es la *línea de estado HTTP* , que incluye el código de estado 200 y el texto de estado *OK* , que indican que la solicitud se procesó correctamente. A esto le siguen varios campos de encabezado HTTP, una línea en blanco y la respuesta HTML.

```
$ telnet www.example.org 80
Trying 2606:2800:220:1:248:1893:25c8:1946...
Connected to www.example.org.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.example.org

HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Thu, 21 Jul 2016 15:56:05 GMT
Etag: "359670651"
Expires: Thu, 28 Jul 2016 15:56:05 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (lga/1318)
Vary: Accept-Encoding
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
```

```
    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in documents. You may
use this
    domain in examples without prior coordination or asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

(Elemento de `style` y líneas en blanco eliminadas de la respuesta de HTML, por brevedad.)

Formato de solicitud básico

En HTTP 1.1, una solicitud HTTP mínima consiste en una línea de solicitud y un encabezado de `Host` :

```
GET /search HTTP/1.1 \r\n
Host: google.com \r\n
\r\n
```

La primera línea tiene este formato:

```
Method Request-URI HTTP-Version CRLF
```

`Method` debe ser un método HTTP válido; uno de [1] [2] :

- `OPTIONS`
- `GET`
- `HEAD`
- `POST`
- `PUT`
- `DELETE`
- `PATCH`
- `TRACE`
- `CONNECT`

`Request-URI` indica el URI o la ruta al recurso que el cliente está solicitando. Puede ser:

- un URI completamente calificado, que incluye esquema, host, puerto (opcional) y ruta; o
- una ruta, en cuyo caso el host debe especificarse en el encabezado del `Host`

`HTTP-Version` indica la versión del protocolo HTTP que utiliza el cliente. Para solicitudes HTTP 1.1, esto siempre debe ser `HTTP/1.1` .

La línea de solicitud finaliza con un retorno de carro: par de avance de línea, generalmente representado por `\r\n` .

Campos de encabezado de solicitud

Los campos de encabezado (generalmente llamados "encabezados") se pueden agregar a una solicitud HTTP para proporcionar información adicional con la solicitud. Un encabezado tiene una semántica similar a los parámetros pasados a un método en cualquier lenguaje de programación que admita tales cosas.

Una solicitud con encabezados `Host` , `User-Agent` y `Referer` puede tener este aspecto:

```
GET /search HTTP/1.1 \r\n
Host: google.com \r\n
User-Agent: Chrome/54.0.2803.1 \r\n
Referer: http://google.com/ \r\n
\r\n
```

Se puede encontrar una lista completa de encabezados de solicitud HTTP 1.1 admitidos en la especificación . Los más comunes son:

- `Host` : la parte del nombre de host de la URL de solicitud (requerida en HTTP / 1.1)
- `User-Agent` : una cadena que representa la solicitud del agente de usuario;
- `Referer` - el URI del cual el cliente fue referido aquí; y
- `If-Modified-Since` : proporciona una fecha que el servidor puede usar para determinar si un recurso ha cambiado e indica que el cliente puede usar una copia en caché si no lo ha hecho.

Un encabezado debe formarse como `Name: Value CRLF` . `Name` es el nombre del encabezado, como `User-Agent` . `Value` es la información que se le asigna, y la línea debe terminar con un CRLF. Los nombres de encabezados no distinguen entre mayúsculas y minúsculas y solo pueden usar letras, dígitos y los caracteres `!#$%&'*+-.^_`|~` (RFC7230 sección 3.2.6 Componentes de valor de campo).

El nombre del campo del encabezado del `Referer` es un error tipográfico para ' `Referer` ', introducido accidentalmente en RFC1945 .

Cuerpos de mensajes

Algunas solicitudes HTTP pueden contener un cuerpo de mensaje. Estos son datos adicionales que el servidor utilizará para procesar la solicitud. Los cuerpos de los mensajes se utilizan con mayor frecuencia en las solicitudes POST o PATCH y PUT, para proporcionar nuevos datos que el servidor debe aplicar a un recurso.

Las solicitudes que incluyen un cuerpo de mensaje siempre deben incluir su longitud en bytes con el encabezado `Content-Length` .

Se incluye un cuerpo de mensaje *después de* todos los encabezados y un doble CRLF. Un ejemplo de solicitud PUT con un cuerpo podría verse así:

```
PUT /files/129742 HTTP/1.1\r\n
Host: example.com\r\n
```

```
User-Agent: Chrome/54.0.2803.1\r\n
Content-Length: 202\r\n
\r\n
This is a message body. All content in this message body should be stored under the
/files/129742 path, as specified by the PUT specification. The message body does
not have to be terminated with CRLF.
```

`HEAD` solicitudes `HEAD` y `TRACE` no deben incluir un cuerpo de mensaje.

Lea Peticiones HTTP en línea: https://riptutorial.com/es/http/topic/2909/peticiones-http

Capítulo 9: Respuestas HTTP

Parámetros

Código de estado	Razón-Frase - Descripción
100	**Continuar** : el cliente debe enviar la siguiente parte de una solicitud de varias partes.
101	**Protocolos de conmutación** : el servidor está cambiando la versión o el tipo de protocolo utilizado en esta comunicación.
200	**OK** - el servidor ha recibido y completado la solicitud del cliente.
201	**Creado** : el servidor ha aceptado la solicitud y ha creado un nuevo recurso, que está disponible bajo el URI en el encabezado `Location` .
202	**Aceptado** : el servidor ha recibido y aceptado la solicitud del cliente, pero aún no ha iniciado o completado el procesamiento.
203	**Información no autorizada** : el servidor está devolviendo datos que pueden ser un subconjunto o superconjunto de la información disponible en el servidor original. Utilizado principalmente por los proxies.
204	**Sin contenido** : se utiliza en lugar de 200 (OK) cuando no hay cuerpo en la respuesta.
205	**Restablecer contenido** : idéntico a 204 (Sin contenido), pero el cliente debe volver a cargar la vista de documento activa.
206	**Contenido parcial** : se usa en lugar de 200 (OK) cuando el cliente solicitó un encabezado de `Range` .
300	**Opciones múltiples** : el recurso solicitado está disponible en varios URI, y el cliente debe redirigir la solicitud a un URI especificado en la lista en el cuerpo del mensaje.
301	**Movido permanentemente** : el recurso solicitado ya no está disponible en este URI, y el cliente debe redirigir esta y todas las solicitudes futuras al URI especificado en el encabezado de la `Location` .
302	**Encontrado** : el recurso reside temporalmente en un URI diferente. Esta solicitud debe redirigirse en la confirmación del usuario al URI en el encabezado de la `Location` , pero las solicitudes futuras no deben modificarse.
303	**Consulte Otros** : muy similar a 302 (Encontrado), pero no requiere la entrada

Código de estado	Razón-Frase - Descripción
	del usuario para redirigir al URI proporcionado. El URI proporcionado se debe recuperar con una solicitud GET.
304	**No modificado** : el cliente envió un encabezado `If-Modified-Since` o similar, y el recurso no se ha modificado desde ese punto; el cliente debe mostrar una copia en caché del recurso.
305	**Usar proxy** : el recurso solicitado debe solicitarse nuevamente a través del proxy especificado en el campo Encabezado de `Location` .
307	**Redireccionamiento temporal** : idéntico a 302 (Encontrado), pero los clientes HTTP 1.0 no admiten 307 respuestas.
400	**Solicitud incorrecta** : el cliente envió una solicitud con formato incorrecto que contiene errores de sintaxis y debe modificar la solicitud para corregirla antes de repetirla.
401	**No autorizado** : el recurso solicitado no está disponible sin autenticación. El cliente puede repetir la solicitud utilizando un encabezado de `Authorization` para proporcionar detalles de autenticación.
402	**Pago requerido** : código de estado reservado y no especificado para que lo utilicen las aplicaciones que requieren suscripciones de usuarios para ver el contenido.
403	**Prohibido** : el servidor entiende la solicitud, pero se niega a cumplirla debido a los controles de acceso existentes. La solicitud no debe repetirse.
404	**No encontrado** : no hay ningún recurso disponible en este servidor que coincida con el URI solicitado. Puede usarse en lugar de 403 para evitar exponer los detalles del control de acceso.
405	**Método no permitido** : el recurso no admite el método de solicitud (verbo HTTP); el encabezado `Allow` enumera los métodos de solicitud aceptables.
406	**No aceptable** : el recurso tiene características que violan los encabezados de aceptación enviados en la solicitud.
407	**Se requiere autenticación de proxy** - similar a 401 (no autorizado), pero indica que el cliente primero debe autenticar con el proxy intermedio.
408	**Tiempo de espera de solicitud** : el servidor esperaba otra solicitud del cliente, pero ninguna se proporcionó dentro de un período de tiempo aceptable.
409	**Conflicto** : la solicitud no se pudo completar porque estaba en conflicto con el estado actual del recurso.

Código de estado	Razón-Frase - Descripción
410	**Desaparecido** : similar a 404 (No encontrado), pero indica una eliminación permanente. No hay dirección de reenvío disponible.
411	**Longitud requerida** : el cliente no especificó un encabezado de `Content-Length` válido, y debe hacerlo antes de que el servidor acepte esta solicitud.
412	**Condición fallida** : el recurso no está disponible con todas las condiciones especificadas por los encabezados condicionales enviados por el cliente.
413	**Solicitud de entidad demasiado grande** : el servidor no puede procesar un cuerpo del mensaje de la longitud que envió el cliente.
414	**El URI de solicitud es demasiado largo** : el servidor rechaza la solicitud porque el URI de solicitud es más largo de lo que el servidor está dispuesto a interpretar.
415	**Tipo de medio no admitido** : el servidor no admite el MIME o el tipo de medio especificado por el cliente y no puede atender esta solicitud.
416	**Intervalo solicitado no satisfactorio** : el cliente solicitó un rango de bytes, pero el servidor no puede proporcionar contenido a esa especificación.
417	**Expectativa fallida** : las restricciones especificadas por el cliente en el encabezado `Expect` que el servidor no puede cumplir.
500	**Error interno del servidor** : el servidor cumplió con una condición o error inesperado que le impide completar esta solicitud.
501	**No implementado** : el servidor no admite la funcionalidad requerida para completar la solicitud. Generalmente se usa para indicar un método de solicitud que no es compatible con *ningún* recurso.
502	**Puerta de enlace incorrecta** : el servidor es un proxy y recibió una respuesta no válida del servidor ascendente al procesar esta solicitud.
503	**Servicio no disponible** : el servidor está bajo una carga alta o está en mantenimiento, y no tiene la capacidad de atender esta solicitud en este momento.
504	**Tiempo de espera de la puerta de enlace** : el servidor es un proxy y no recibió una respuesta del servidor ascendente de manera oportuna.
505	**Versión HTTP no admitida** : el servidor no admite la versión del protocolo HTTP con el que el cliente realizó su solicitud.

Examples

Formato de respuesta basico

Cuando un servidor HTTP recibe una solicitud HTTP bien formada, debe procesar la información que contiene la solicitud y devolver una respuesta al cliente. Una simple respuesta HTTP 1.1, puede parecerse a cualquiera de las siguientes, generalmente seguida de una cantidad de campos de encabezado, y posiblemente un cuerpo de respuesta:

```
HTTP/1.1 200 OK \r\n
```

```
HTTP/1.1 404 Not Found \r\n
```

```
HTTP/1.1 503 Service Unavailable \r\n
```

Una simple respuesta HTTP 1.1 tiene este formato:

```
HTTP-Version Status-Code Reason-Phrase CRLF
```

Al igual que en una petición, `HTTP-Version` indica la versión del protocolo HTTP en uso; para HTTP 1.1, esta debe ser siempre la cadena `HTTP/1.1` .

`Status-Code` es un código de tres dígitos que indica el estado de la solicitud del cliente. El primer dígito de este código es la *clase de estado* , que coloca el código de estado en una de las 5 categorías de respuesta [1] :

- `1xx` **Informativo** : el servidor ha recibido la solicitud y el proceso continúa
- `2xx` **Success** - el servidor ha aceptado y procesado la solicitud
- **Redireccionamiento** `3xx` : es necesaria una acción adicional por parte del cliente para completar la solicitud
- `4xx` **Errores de cliente** : el cliente envió una solicitud con un formato incorrecto o que no se pudo cumplir
- `5xx` **Errores del servidor** : la solicitud era válida, pero el servidor no puede cumplirla en este momento

`Reason-Phrase` es una breve descripción del código de estado. Por ejemplo, el código `200` tiene una frase de razón de `OK` ; El código `404` tiene una frase de `Not Found` . Una lista completa de frases de motivos está disponible en Parámetros, a continuación, o en la especificación HTTP .

La línea termina con un retorno de carro: par de avance de línea, generalmente representado por `\r\n` .

Encabezados adicionales

Al igual que una solicitud HTTP, una respuesta HTTP puede incluir encabezados adicionales para modificar o aumentar la respuesta que proporciona.

Una lista completa de los encabezados disponibles se define en §6.2 de la especificación . Los encabezados más utilizados son:

- `Server` , que funciona como un encabezado de solicitud de `User-Agent` para el servidor;
- `Location` , que se usa en las respuestas de estado 201 y 3xx para indicar un URI para redirigir a; y
- `ETag` , que es un identificador único para esta versión del recurso devuelto para permitir a los clientes almacenar en caché la respuesta.

Los encabezados de respuesta vienen después de la línea de estado y, al igual que con los encabezados de solicitud, se forman como tales:

```
Name: Value CRLF
```

`Name` proporciona el nombre del encabezado, como `ETag` o `Location` , y `Value` proporciona el valor que el servidor está configurando para ese encabezado. La línea termina con un CRLF.

Una respuesta con encabezados podría verse así:

```
HTTP/1.1 201 Created \r\n
Server: WEBrick/1.3.1 \r\n
Location: http://example.com/files/129742 \r\n
```

Cuerpos de mensajes

Al igual que con los cuerpos de solicitud , las respuestas HTTP pueden contener un cuerpo de mensaje. Esto proporciona datos adicionales que el cliente procesará. En particular, 200 respuestas OK a una solicitud GET bien formada siempre deben proporcionar un cuerpo de mensaje que contenga los datos solicitados. (Si no hay ninguno, 204 No Content es una respuesta más apropiada).

Se incluye un cuerpo de mensaje después de todos los encabezados y un doble CRLF. En cuanto a las solicitudes, su longitud en bytes debe darse con el encabezado `Content-Length` . Una respuesta exitosa a una solicitud GET, por lo tanto, podría verse así:

```
HTTP/1.1 200 OK\r\n
Server: WEBrick/1.3.1\r\n
Content-Length: 39\r\n
ETag: 4f7e2ed02b836f60716a7a3227e2b5bda7ee12c53be282a5459d7851c2b4fdfd\r\n
\r\n
Nobody expects the Spanish Inquisition.
```

Lea Respuestas HTTP en línea: https://riptutorial.com/es/http/topic/3077/respuestas-http

Creditos

S. No	Capítulos	Contributors
1	Empezando con HTTP	Community, DaSourcerer, Kornel, Peter Hilton
2	Autenticación	DaSourcerer, Peter Hilton, Stefan Kögl
3	Cachear las respuestas HTTP	DaSourcerer, Kornel
4	Codificaciones de respuesta y compresión.	Jeff Bencteux, Peter Hilton
5	Códigos de estado HTTP	ArtOfCode, DaSourcerer, Deltik, Kornel, Lex Li, mnoronha, Peter Hilton, Rptk99, Sender, Xevaquor
6	HTTP para APIs	ArtOfCode, mnoronha, Peter Hilton, Roman Vottner
7	Origen cruzado y control de acceso	ArtOfCode
8	Peticiones HTTP	artem, ArtOfCode, Jeff Bencteux, Peter Hilton
9	Respuestas HTTP	ArtOfCode, Jeff Bencteux, Peter Hilton