Objeto Respuesta

Tiempo estimado: 25min

En una aplicación Express, tan importante como conocer el objeto de petición HTTP es conocer el objeto de respuesta HTTP. A través del cual redactar la respuesta a remitir al cliente.

Para comenzar, se introduce el concepto de mensaje de respuesta. Uno de los dos tipos de mensajes HTTP. Recordemos que el otro es el mensaje de petición. A continuación, se describe detalladamente la línea de estado, algunos campos de cabecera y el cuerpo del mensaje. Finalmente, se presenta el objeto Response con el que redactar el mensaje de respuesta a remitir al usuario.

Al finalizar la lección, el estudiante sabrá:

- Qué es el mensaje de respuesta.
- Cómo redactar la respuesta.
- Qué es el estado, el código de estado y la descripción de estado.
- Cuáles son los campos de cabecera más importantes relacionados con el contenido remitido en un mensaje, ya sea de petición o de respuesta.

Introducción

Un mensaje de respuesta (*response message*) es aquel que remite el servidor al cliente como contestación a su petición. Al igual que los mensajes de petición, es de texto y su sintaxis es la siguiente:

```
Línea de estado
Campos de cabecera
Línea en blanco
Cuerpo del mensaje
```

Línea de estado

Toda respuesta debe comenzar con una línea de estado (*status line*). Esta línea tiene un objeto similar a la línea de petición: indicar la especificación HTTP en la que se encuentra redactado el mensaje, así como el estado en formato numérico y textual:

```
Versión-HTTP Código-de-estado Texto-de-estado
```

Primero, se indica la versión del protocolo HTTP. Le sigue el estado (*status*), un valor que representa cómo acabó el procesamiento de la petición HTTP de la que este mensaje es su respuesta. Tiene un valor numérico y otro textual. El valor numérico es una secuencia de tres dígitos y se conoce formalmente como código de estado (status code) o código de respuesta (*response code*). El valor textual no es más que una breve descripción del estado, la cual se conoce como descripción de estado (*status description*). Un ejemplo de estado es 404 Not Found.

Los estados se clasifican en cinco categorías, atendiendo al resultado del procesamiento: 1xx o informativos; 2xx o de éxito; 3xx o de redireccionamiento; 4xx o de error cliente; ó 5xx o de error servidor. A continuación, se presenta los más frecuentemente usados.

Estados 1xx o informativos

Los códigos de respuesta 1xx o informativos agrupan aquellos estados que no pertenecen a ninguno de los otros grupos. Disponemos de los siguientes códigos:

- 100 Continue. Mediante este código, el servidor le indica al cliente que las cabeceras de su petición HTTP han llegado y han sido aceptadas y que continúe con el resto del mensaje.
- 101 Switching Protocols. En ocasiones, el cliente puede solicitar un cambio de versión de protocolo.

Mediante este código, el servidor indica que está de acuerdo en el cambio.

- 102 Processing. El servidor ha aceptado la petición, pero todavía la está procesando. Este tipo de mensaje lo envía cuando lleva mucho tiempo procesando una petición y desea informar al cliente que aún no ha terminado, para que no se impaciente.
- 103 Checkpoint. Mediante este código el servidor le informa al cliente que reanuda una petición POST o PUT que se abortó anteriormente.

Estados 2xx o de éxito

Mediante el grupo de códigos 2xx, el servidor informa al cliente que su petición ha sido recibida, aceptada y, en algunos casos, incluso completada:

- 200 OK. Con este tipo de respuesta, el servidor está indicándole al cliente que su petición ha sido aceptada, procesada y que en el cuerpo se encuentra la copia del recurso solicitado. Es la respuesta estándar a las peticiones GET. Además, con este código, indica que el contenido de recurso es original, de un servidor válido.
- 201 Created. Respuesta con la que el servidor indica que la petición ha sido procesada y ha creado el recurso solicitado por el cliente. Generalmente, se utiliza como respuesta de peticiones POST, recordemos, aquellas que suelen llevar asociada la creación de un nuevo recurso.
- 202 Accepted. La petición ha sido aceptada y analizada, pero todavía su procesamiento no ha finalizado, ya sea porque no se ha comenzado todavía o está en ello ahora mismo.
- 203 Non-Authoritative Information. Es similar al estado 200 OK, pero con este código indicamos que el servidor no es una fuente válida del recurso, es decir, procede de otro servidor, el cual no se reconoce como origen autorizado del recurso.
- 204 No Content. La petición se ha procesado con éxito, pero la respuesta no dispone de contenido. Es muy similar al 200, pero el 200 siempre lleva contenido y el 204 nunca.
- 205 Reset Content. Es similar al 204, pero con el 205 el servidor solicita al cliente que actualice la página del usuario.
- 206 Partial Content. Similar al 200 OK, pero le indica que el mensaje sólo contiene parte del contenido. Con un 200, el mensaje siempre contiene la copia completa del recurso. Con un 206, sólo aquellos datos que ha solicitado el cliente.

Estados 3xx o de redireccionamiento

Estos los veremos más adelante en la lección de redireccionamiento HTTP.

Estados 4xx o de error cliente

Con los códigos 4xx, el servidor indica que no puede procesar la petición HTTP por alguna causa relacionada con la petición como, por ejemplo, por no existir el recurso solicitado:

- 400 Bad Request. Petición HTTP mal redactada, por ejemplo, por un error de sintaxis.
- 401 Unauthorized. En este caso, el recurso requiere que el usuario se haya autenticado. O bien no lo ha hecho o lo ha hecho incorrectamente.
- 403 Forbidden. El usuario no tiene derecho para acceder al recurso o para realizar la acción especificada mediante el método o verbo indicado en la petición.
- 404 Not Found. El recurso solicitado no existe.
- 405 Method Not Allowed. El método indicado en la petición no se puede aplicar al recurso.
- 406 Not Acceptable. El servidor no puede proporcionar el recurso al cliente en ninguno de los formatos indicados en los campos de cabecera Accept* remitidos en la petición.
- 407 Proxy Authentication Required. El usuario debe autenticarse ante el *proxy* indicado en el campo de

cabecera Location. Una vez lo haya hecho, puede remitir de nuevo la petición.

- 408 Request Timeout. El servidor esperaba que el cliente respondiera y ha transcurrido el tiempo de espera máximo.
- 409 Conflict. El servidor no pudo realizar la acción solicitada debido al estado en que se encuentra el recurso actualmente. El servidor debería adjuntar un mensaje descriptivo del problema en el cuerpo de la respuesta.
- 410 Gone. El servidor ya no es responsable de servir el recurso solicitado ni lo será nunca más. Recomienda que el cliente suprima el enlace de su aplicación o base de datos.
- 411 Length Required. El servidor rechaza la petición por no disponer del campo de cabecera Content-Length.
- 412 Precondition Failed. El servidor no es capaz de cumplir algunas de las condiciones indicadas en la petición.
- 413 Request Entity Too Large. El servidor no acepta peticiones de ese tamaño.
- 414 Request-URI Too Long. El servidor no acepta peticiones con URIs de ese tamaño.
- 415 Unsupported Media Type. El servidor no acepta peticiones del tipo de contenido indicado.
- 417 Expectation Failed. El servidor no es capaz de cumplir lo indicado en el campo de cabecera Expect de la petición.
- 423 Locked. El recurso solicitado se encuentra bloqueado actualmente.
- 451 Unavailable for Legal Reasons. El recurso ya no está disponible por razones legales.

Estados 5xx o de error servidor

Cuando se produce un error en el servidor, ajeno a la petición y al cliente, el servidor debe responder con un código 5xx:

- 500 Internal Server Error. Se ha producido un error durante el procesamiento de la petición, ajeno al servidor web pero en la máquina servidora. Por ejemplo, se remite cuando el motor de plantillas de la aplicación falla con una plantilla.
- 501 Not Implemented. El servidor no tiene implementada la funcionalidad requerida en la petición.
- 502 Bad Gateway. Este tipo de código lo remiten los *proxies* cuando reciben una respuesta del servidor web mal redactada, por lo que no puede llevar a cabo su función correctamente con la petición del cliente.
- 503 Service Unavailable. El servidor no puede responder porque tiene exceso de carga o se está llevando a cabo tareas de mantenimiento. En este caso, se recomienda que se remita un campo de cabecera Retry-After para indicarle al cliente o usuario cuándo podría intentarlo de nuevo.
- 504 Gateway Timeout. Remitido por *proxies*. Indica al cliente que el servidor web no responde.
- 505 HTTP Version Not Supported. Versión HTTP no soportada por el servidor.
- 507 Insufficient Storage. El servidor no puede almacenar el recurso remitido por el cliente por falta de espacio.

Cuerpo del mensaje

El cuerpo del mensaje (*message body*) es la parte del mensaje que contiene la copia del recurso solicitado. Cuando el mensaje de respuesta lleva consigo contenido, lo adjunta al final del mensaje. Separándolo mediante una línea en blanco (CRLF) de los campos de cabecera. Formalmente, la especificación HTTP usa el término entidad (*entity*) para referirse a un conjunto de datos ubicado en el cuerpo del mensaje.

Campos de cabecera relacionados con el contenido

A continuación, se presenta los campos de cabecera más utilizados con el procesamiento del contenido de los mensajes. Tanto de petición como de respuesta.

Campo de cabecera Content-Type

El campo de cabecera Content-Type se puede adjuntar en la cabecera de los mensajes de petición y de respuesta. E indica la naturaleza o tipo de los datos de la entidad ubicada en el cuerpo. Se utiliza básicamente para indicarle al otro extremo qué tipo de recurso se adjunta: un archivo de texto, un documento HTML, una imagen, un vídeo, etc. De esta manera, el receptor del mensaje dispone de más información para procesar mejor el contenido. Así, por ejemplo, si se trata de un archivo PDF, el navegador podría abrir Adobe Reader para visualizar el contenido al usuario.

La sintaxis de este campo es:

```
Content-Type: tipo/subtipo (; Parámetro)*
```

El tipo de contenido se indica mediante tipos MIME. MIME (*Multipurpose Internet Mail Extensions*, Extensiones Multipropósito de Correo de Internet) es un formato estandarizado para comunicar y transportar datos en un mensaje.

Un tipo MIME (*MIME type*) representa un formato de datos. Atendiendo a la naturaleza de los datos, los tipos MIME se clasifican en text, image, audio, video o application. A su vez, estos tipos se dividen en subtipos (*subtypes*) que especifican más claramente los datos. Por ejemplo, text/html se utiliza para indicar un documento HTML; text/plain, texto claro sin formato; image/jpeg, una imagen JPEG; application/msword, un documento Word; etc.

Además, disponemos de parámetros de tipo (*type parameters*) con los que proporcionar información adicional sobre los datos. Se adjuntan después del tipo MIME separado por un punto y coma (;). Si disponemos de varios parámetros, su orden no es significativo. Estos parámetros presentan la siguiente sintaxis:

```
Nombre=Valor
```

Tipo de contenido text

El tipo de contenido text indica que la entidad transportada en el mensaje es texto. Distinguimos básicamente los siguientes:

- text/plain. Texto plano sin formato.
- text/html. Documento HTML.
- text/xml. Documento XML.
- text/css. Documento de hoja de estilo CSS.

A través del parámetro opcional charset podemos indicar el conjunto de caracteres en los que se encuentra formateado el texto.

Tipo de contenido image

El tipo de contenido image se reserva para imágenes. Las imágenes más transmitidas son: image/jpeg, image/bmp, image/gif e image/png.

Tipo de contenido audio

Para transmitir audio, se utiliza el tipo de contenido audio. Mediante el subtipo seremos más explícitos. He aquí algunos ejemplos: audio/mpeg, audio/mid, audio/aiff y audio/x-ms-wma.

Tipo de contenido video

Si lo que estamos transmitiendo es vídeo, usaremos el tipo de contenido video. Siendo más precisos mediante el subtipo como, por ejemplo, video/mpeg y video/qt.

Tipo de contenido application

El tipo de contenido application se utiliza para indicar contenido que no es ninguno de los anteriores. He aquí

algunos ejemplos: application/octet-stream, datos en binario; application/pdf, archivo PDF; etc.

Tipos de contenido no estandarizados

Si es necesario transmitir datos en un formato no estandarizado, podemos crear nuestro propio subtipo . Sólo hay que mantener un convenio: el subtipo debe ubicarse dentro del tipo más adecuado y su nombre debe comenzar por una x seguida de un guión (x-). Ejemplo: application/x-zip-compressed.

Campos de cabecera Content-Language

Algunas aplicaciones webs tienen la capacidad de saber en qué idioma se encuentra escrito un determinado recurso de texto. Ya sea porque esta información se encuentra en el nombre del archivo, porque se almacena en una base de datos o porque la aplicación es así de lista. Es más, algunos recursos se pueden disponer en varios idiomas, indicando el cliente mediante el campo de cabecera Accept-Language su preferencia de idiomas.

Cuando se da el caso, la aplicación puede indicar el idioma del contenido remitido mediante el campo de cabecera Content-Language:

```
Content-Language: idioma
```

Ejemplo:

```
Content-Language: es
```

Campo de cabecera Content-Length

El campo de cabecera Content-Length es otro campo que puede adjuntarse tanto en las solicitudes como en las respuestas que transporten contenido. Se utiliza para indicar el tamaño, en *bytes*, del cuerpo del mensaje. Su sintaxis es la siguiente:

```
Content-Length: tamaño
```

Campo de cabecera Last-Modified

Los servidores pueden adjuntar, si lo desean, un campo Last-Modified para indicar la fecha de la última modificación del recurso. Cuando un cliente recibe una respuesta con esta cabecera, puede entonces cachear la copia del recurso remitida. Y si el usuario desea acceder a él de nuevo, pueden remitir un mensaje If-Modified-Since indicando que sólo vuelva a remitirle el recurso si ha cambiado desde la fecha indicada. De esta manera, se reduce el tráfico de red y en muchas ocasiones la latencia.

Otros campos de cabecera

A continuación, presentamos algunos campos de cabecera muy utilizados en las respuestas.

Campo de cabecera Server

El campo de cabecera Server es similar al User-Agent de las peticiones, pero proporcionando información sobre el servidor. Por seguridad, no se recomienda adjuntar este campo.

Campo de cabecera Retry-After

Mediante el campo de cabecera Retry-After, el servidor le indica al cliente que ahora no puede atender su petición pero que cree que pasado el tiempo indicado podrá hacerlo sin problemas:

```
Retry-After: Fecha-HTTP|Segundos
```

Se puede indicar la fecha a partir de la cual cree que podrá o un intervalo de tiempo en segundos.

Objeto response

Como el objeto de un servidor o aplicación web es procesar peticiones HTTP y responder a los remitentes, Express proporciona el parámetro response o simplemente res de los controladores para representar la respuesta en curso. En las funciones de *middleware* normales, este parámetro es el segundo. Consiste en una instancia de la clase Response que contiene propiedades y métodos a través de los cuales escribir la respuesta.

Línea de estado

Recordemos que la línea de estado de la respuesta contiene información del estado de la respuesta y de la versión HTTP usada en la redacción del mensaje. En nuestro caso, sólo hay que preocuparse del valor numérico de respuesta, esto es, del código de estado. Su descripción la fija Express atendiendo al código. Cuando no indicamos nada, la aplicación fija automáticamente el código 200 a la respuesta. Si es necesario establecer otro, hay que invocar el método status() del objeto respuesta:

```
status(code) : Response
```

Parámetro	Tipo de datos	Descripción
code	number	Código de estado o respuesta.

El método devuelve la propia respuesta para permitirnos encadenar otras llamadas en la misma proposición. He aquí un ejemplo ilustrativo:

```
res.status(404).send("Recurso no encontrado.");
```

También es posible usar el método sendStatus(), mediante el cual se fija el código de estado y se envía en el cuerpo la descripción del estado:

```
sendStatus(code)
```

Lo siguiente:

```
res.sendStatus(404);
```

Es lo mismo que:

```
res.status(404).send("Not Found");
```

Recordemos que en ambos casos, Express fija la descripción del estado en la línea de estado a partir del código indicado.

Cabecera del mensaje

Para añadir campos de cabecera al mensaje de respuesta, hay que utilizar el método set():

```
set(field)
set(field, value)
set(fields)
```

Parámetro	Tipo de datos	Descripción
field	string	Nombre del campo de cabecera.
value	string	Valor del campo de cabecera.
fields	object	Campos de cabecera a fijar. Cada propiedad es un campo de cabecera cuyo valor es el indicado.

He aquí unos ejemplos ilustrativos:

```
res.set("Content-Type", "text/html");
res.set({
  "Content-Type": "text/html",
  "Server": "My Express app"
});
```

Es muy importante no olvidar nunca que los campos de cabecera hay que fijarlos antes de que hayamos enviado contenido. Como la cabecera va antes que el cuerpo, cuando invocamos un método de envío de contenido como, por ejemplo, write(), send(), json() o sendFile(), la aplicación envía la respuesta y por ende los campos de cabecera que hayamos registrado. Una vez enviado contenido, ya no podremos fijar nuevas cabeceras.

Si en algún momento necesitamos conocer el valor de un determinado campo de cabecera de la respuesta, podemos usar el método get():

```
get(field) : object
```

Cuerpo del mensaje

Para añadir cuerpo al mensaje podemos hacerlo básicamente de dos formas, poco a poco o todo de una vez.

Métodos write() y end()

Para añadir cuerpo poco a poco, hay que utilizar dos métodos, write() y end(). Con write(), lo que hacemos es añadir nuevo contenido; mientras que con end(), le indicamos a Express que hemos finalizado y debe dar por terminada la respuesta.

La sintaxis de write() es:

```
write(chunk)
write(chunk, encoding)
write(chunk, encoding, callback)
```

Parámetro	Tipo de datos	Descripción
chunk	string o Buffer	Contenido a enviar.
encoding	string	Codificación en la que enviarlo. Valor predeterminado: utf8.
callback	function	Función a llamar cuando se haya enviado el contenido.

Una vez enviado el contenido, lo último que queda es informar de que hemos terminado. Para ello, recordemos, tenemos el método end():

```
end()
end(data)
end(data, encoding)
end(data, encoding, callback)
```

Parámetro	Tipos de datos	Descripción
data	string o Buffer	Último contenido a enviar.
encoding	string	Codificación del último contenido a enviar. Valor predeterminado: utf8.
callback	function	Función a llamar cuando se haya terminado.

A continuación, se muestra un ejemplo de cómo enviar contenido en varios bloques:

```
res.write("Primera parte del contenido");
res.write("Segunda parte del contenido");
res.end("Última parte del contenido");
```

Método send()

Cuando podemos enviar todo el contenido de una única tacada, podemos hacerlo mediante el método send(), el cual tiene implícito la invocación a write() y end():

```
send(data)
```

Parámetro	Tipos de datos	Descripción
data	string, object, Buffer	Contenido a enviar.

La ventaja de usar send() es que este método fija el valor del campo de cabecera Content-Length automáticamente. Algo que no sucede cuando se usa los métodos write() y end(), en los que es necesario hacerlo explícitamente antes de la primera invocación a write().

Cuando el parámetro data es un Buffer, el método fija como tipo de contenido application/octet-stream. Si es una cadena, text/html. Y si es un objeto, application/json. Si fuera otro, no olvidar fijar el campo de cabecera Content-Type antes de la invocación a send().

Ejemplo ilustrativo:

```
res.status(404).send("Recurso no encontrado.");
```

Método json()

Para enviar un objeto JSON, podemos utilizar el método json(), el cual obtiene una representación del objeto JavaScript pasado en formato JSON, lo envía mediante write() y end() y además fija el campo de cabecera Content-Type a application/json:

```
json(obj)
```

Parámetro	Tipo de datos	Descripción
obj	object	Objeto JavaScript a remitir como cuerpo.

Veamos un ejemplo:

```
res.json({
  band: "The Wild Swans",
  origin: "Liverpool, England"
});

//similar a:
res.set("Content-Type", "application/json");

res.write(JSON.stringify({
  band: "The Wild Swans",
  origin: "Liverpool, England"
}));

res.end();
```

Método sendFile()

Para facilitar el envío de archivos de disco y/o listados de entradas de un directorio, la respuesta proporciona el método sendFile():

```
sendFile(path)
sendFile(path, options)
sendFile(path, options, callback)
```

Parámetro	Tipos de datos	Descripción
path	string	Ruta al archivo o directorio a enviar como contenido. Debe ser absoluta a menos que se indique la opción root, en cuyo caso debe ser relativa a root.
options	object	Opciones de envío: • maxAge (number). Milisegundos que puede cachear el cliente y/o *proxy* el archivo. Valor predeterminado: 0. • root (string). Directorio raíz a usar para formar la ruta absoluta con la indicada en path. • lastModified (boolean). ¿Añadir el campo de cabecera Last-Modified? Valor predeterminado: true. • headers (object). Campos de cabecera a añadir a la respuesta. • dotFiles (string). ¿Enviar, en el listado del directorio, los archivos que comienzan por un punto? allow, sí; deny, no; e ignore, tampoco. Valor predeterminado: ignore. • acceptRanges (boolean). ¿Añadir el campo de cabecera Accept-Ranges a la respuesta? Valor predeterminado: true. • cacheControl (boolean). ¿Añadir el campo de cabecera Cache-Control a la respuesta? Valor predeterminado: true.
callback	function	Función a invocar cuando la transferencia del archivo haya terminado: fn(err).

Este método intenta fijar el mejor tipo de contenido para el archivo remitido, si no se fija explícitamente. Para ello, se sirve de la extensión del archivo. Así, por ejemplo, si es .json, remitirá application/json.

A continuación, un ejemplo ilustrativo:

```
res.sendFile("bands/close-lobsters.json", {
  root: __dirname,
  headers: {
    "Content-Type": "application/json"
  },
  cacheControl: true,
```

```
  maxAge: 60000,
  lastModified: true
}, function(err) {
  console.log("Error al enviar archivo bands/close-lobsters.json.");
  res.status(err.status).end();
)};
```

