API Fetch

Tiempo estimado: 10min

En este punto del curso, ya sabemos cómo usar la arquitectura Flux para desarrollar aplicaciones React profesionales con acceso a datos. Pero las prácticas las hemos realizado siempre con acceso síncrono. Con datos almacenados en la propia aplicación. Habitualmente, el acceso es asíncrono y los almacenes usan servicios webs REST para el acceso a los datos. En esta lección, explicamos la API Fetch, la preferida para el acceso del almacén al servidor web para la lectura/escritura del origen de datos que tiene asociado.

La lección comienza presentando la API Fetch y quién la proporciona. A continuación, se expone detenidamente la función fetch(), la que utilizarán los almacenes para acceder a su origen de datos asíncronamente. Después de esto, se presenta el objeto respuesta, donde se encuentra los datos remitidos por el servidor. Y finalmente, cómo seguir el redireccionamiento mediante Fetch.

Al finalizar la lección, el estudiante sabrá:

- Para qué se usa la API Fetch.
- Cómo usar la API Fetch.
- Quién proporciona la API Fetch.

Introducción

Fetch es una API para el acceso a recursos mediante JavaScript y HTTP desde una aplicación web cliente. Consiste en una API asíncrona *que implementan los navegadores* y podemos usar para acceder a recursos del servidor web como, por ejemplo, los datos que proporciona o modifica un almacén Flux. Se encuentra estandarizada, encostrándose su especificación en fetch.spec.whatwg.org.

Esta interfaz viene de fábrica con los navegadores webs. Es muy importante recordarlo, por lo que no hay que añadir ninguna dependencia en el archivo package.json. Para realizar una consulta HTTP basta con usar la función global fetch().

Función fetch

Para realizar consultas HTTP desde una aplicación web cliente, podemos usar la función fetch(), la cual recibe la información de la solicitud HTTP y devuelve una promesa a través de la cual acceder al objeto resultado:

```
function fetch(input) : Promise<Response>
function fetch(input, init) : Promise<Request>
```

Parámetro	Tipo de datos	Descripción
input	string o Request	URL del recurso a solicitar.
init	object	Opciones de consulta: • method (string). Método HTTP a usar. • headers (object). Cabeceras HTTP a adjuntar. • body (object). Datos a adjuntar en el cuerpo. • mode (string). Modo a usar: cors, no-cors o same-origin. • credentials. Credenciales a usar. • cache (string). Configuración de caché: default, no-store, reload, no-cache, force-cache u only-if. • redirect (string). Qué hacer si la respuesta es un redireccionamiento: follow, error o manual. • referrer (string). ¿Indicar la cabecera Referrer? no-referrer, client o una URL.

Copyright © 2016 nodemy.com. Reservados todos los derechos.

- referrerPolicy (string). Valor a adjuntar en la cabecera Referrer: no-referrer, no-referrer-when-downgrade, origin, origin-when-cross-origin o unsafe-url.
- integrity. Valor de integridad del mensaje.

He aquí un ejemplo ilustrativo:

```
fetch(new Request("/data/archivo.json")).then(function(res) {
  …
});
```

Como se puede observar, para acceder al objeto que describe la respuesta recibida del servidor, se utiliza el método then() de la promesa, el cual espera una función a invocar para pasarnos, así, la respuesta a procesar.

Objeto respuesta

La función fetch() devuelve una promesa con la que acceder al objeto que describe el mensaje respuesta remitido por el servidor. Este objeto se pasa como argumento en las funciones que se pasan al método then() de la promesa.

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. El código de estado se puede consultar mediante las propiedades status y statusText. La primera indica el valor numérico; y la segunda, la descripción. Si lo deseamos, podemos usar la propiedad booleana ok para saber si el código de estado pertenece al grupo 2xx.

Cabeceras de mensaje

Las cabeceras de mensaje proporcionan información adicional del mensaje. Se pueden acceder mediante la propiedad headers del objeto respuesta. Esta propiedad proporciona métodos con los que acceder a la información de cabecera.

Para conocer los campos de cabecera remitidos en el mensaje, se puede usar el método keys() que devuelve un iterador, donde cada elemento es el nombre de un campo:

```
keys() : iterator
```

Si necesitamos saber si el mensaje adjunta un determinado campo de cabecera, podemos usar has():

```
has(name) : boolean
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre del campo de cabecera.

Mientras que para acceder al valor de un determinado campo de cabecera, los métodos get() y getAll():

```
get(name) : string
getAll(name) : string[]
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre del campo de cabecera.

Si el campo de cabecera presenta varios valores, mediante get() se obtendrá el primero de ellos, mientras que con getAll() todos ellos. Si sólo presenta uno, ambos métodos devolverán lo mismo, salvo que el segundo lo hará en un *array*.

Para acceder a los distintos campos de cabecera mediante pares clave-valor, se puede utilizar el método iterador entries(), el cual, para cada iteración, devuelve un *array*, donde el primer elemento es el nombre del campo y el segundo su valor:

```
entries() : iterator
```

Copyright © 2016 nodemy.com. Reservados todos los derechos.

Cuerpo del mensaje

Recordemos que el cuerpo del mensaje (*message body*) es la parte del mensaje que contiene la copia del recurso solicitado. Para saber si el mensaje de respuesta tiene cuerpo, se puede consultar la propiedad booleana bodyUsed.

Cuando un almacén usa fetch() para acceder asíncronamente a un recurso web, generalmente, lo que recibe es contenido de tipo JSON o texto.

Cuerpos en formato JSON

Si sabemos que el cuerpo es un objeto en formato JSON, podemos obtenerlo en forma de objeto JavaScript mediante el método json() de la respuesta:

```
json() : Promise<object>
```

El método devuelve una promesa con la que obtener el objeto JavaScript. Ejemplo:

```
res.json().then(function(obj) {
  …
});
```

Cuerpos en formato texto

Si lo que transporta el cuerpo es un texto, podemos acceder a él mediante el método text(), el cual devuelve una promesa con la que obtenerlo:

```
text() : Promise<string>
```

Ejemplo:

```
res.text().then(function(txt) {
  …
})
```

Redireccionamiento

Es posible que el servidor devuelva un estado de redireccionamiento. Para estos casos, podemos indicarle a fecth() cómo debe actuar si la respuesta es de redireccionamiento. Para ello, usaremos la opción redirect que espera una cadena de texto con uno de los siguientes valores:

- follow. Sigue automáticamente el enlace, es decir, realiza una consulta automática del recurso redirigido.
- error. Propaga un error si se recibe un mensaje de respuesta de redireccionamiento.
- manual. No seguir el enlace. Si hay que hacerlo, lo haremos manualmente mediante otra nueva consulta.

Veamos un ejemplo mediante el cual configurar que el propio fetch() siga los enlaces automáticamente:

```
fetch(path, {redirect: "follow"}).this(function(res) {
  …
});
```

Cuando se configura el seguimiento automático, se puede utilizar la propiedad booleana redirected del objeto respuesta para saber si fetch() ha tenido que seguir algún redireccionamiento.

Copyright © 2016 nodemy.com. Reservados todos los derechos.