Tutorial de fetch, el API de Javascript para las comunicaciones asíncronas basado en promesas. Es la manera más sencilla y práctica de hacer Ajax con código nativo en el navegador.

Fetch es un nuevo API (ya no tan nuevo) para el acceso a recursos del servidor de manera asíncrona, basado en promesas. Es básicamente la nueva interfaz para realizar funcionalidades Ajax con Javascript, que ya podemos usar

solo con la instalación del correspondiente polyfill. Esto es un tema de otro artículo: Polyfill para Fetch.

En este tutorial de fectch veremos unos primeros ejemplos sencillos con Fetch y más adelante hablaremos de usos más avanzados, como el acceso a las API REST. Lo que no vamos a explicar es el uso de las promesas de Javascript ES6, ya que ha sido materia de estudio de artículos anteriores. Tampoco vamos a tratar todavía sobre el uso de sus Polyfill y sus diferentes alternativas para compatibilidad, que analizaremos en breve en futuros artículos.

Método fetch

El método fetch() depende directamente del objeto window del navegador. Su uso más simple consiste en pasarle una URL, cuyo contenido se traerá el cliente web de manera asíncrona.

```
fetch('test.txt')
  .then(ajaxPositive)
  .catch(showError);
```

Como sabes, then() nos servirá para definir la función que se encargará de realizar acciones en el caso positivo y catch() para definir una función con código a ejecutar en el caso negativo.

Tratamiento del error con catch()

Comenzamos por ver el código a ejecutar en el caso negativo, que es más sencillo. Simplemente definimos una función que recibirá como parámetro el motivo del error.

el archivo .html en un servidor web y acceder a él mediante http://.

Si la página desde la que se hace el fetch se accede desde file:// observaremos como la llamada Ajax nos da un error y el flujo de ejecución de nuestro código se va por la parte del catch.

Tratamiento del caso positivo con then()

Cuando se ejecuta la solicitud de manera correcta podemos escribir el código del caso positivo en la función que asignamos al then() asociado al método fetch().

El caso positivo siempre nos devolverá una respuesta del servidor, sobre la que se pueden realizar varias cosas, básicamente saber los detalles derivados del protocolo HTTP de la respuesta y acceder al contenido que el propio servidor nos ha enviado.

```
function ajaxPositive(response) {
  console.log('response.ok: ', response.ok);
  if(response.ok) {
    response.text().then(showResult);
  } else {
    showError('status code: ' + response.status);
    return false;
  }
}
```

Nota: Fíjate que la función ajaxPositive() fue enviada como parámetro en el then() asociado al fetch. Primer código de este artículo.

La función que asocias al "then" del método text() de la respuesta recibe como parámetro el contenido de texto recibido por el servidor. En nuestro caso habíamos asociado "showResult" y en el código de esa función es donde podremos ver cómo usar el texto de la respuesta.

```
function showResult(txt) {
  console.log('muestro respuesta: ', txt);
}
```

Mejora del código mediante arrow functions

Ejemplo completo de Ajax con fetch

Para entender mejor un uso completo de uso de fetch encontrarás ahora el código del ejercicio que hemos usado como base para escribir este artículo.

Por simplificarlo bastante, todos los mensajes de respuesta los enviamos a la consola, por lo que tendrás que abrirla para observar la salida del programa.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    .catch(showError);
});

function ajaxPositive(response) {
  console.log('response.ok: ', response.ok);
  if(response.ok) {
    response.text().then(showResult);
  } else {
    showError('status code: ' + response.status);
  }
}

function showResult(txt) {
  console.log('muestro respuesta: ', txt);
```

Acceso a un API REST mediante fetch

Por supuesto, con fetch podemos acceder a los servicios web y recuperar datos que nos ofrece un API REST. El mecanismo es muy similar a lo que hemos aprendido en este artículo hasta el momento, con la única diferencia que tendremos que comprobar la respuesta de nuestro API para saber si la conexión ha sido satisfactoria.

Con fetch podemos hacer todo tipo de conexiones HTTP. Vamos a ver un ejemplo sobre una solicitud a un API mediante GET, para recuperar datos.

```
fetch('https://randomuser.me/api/?results=10')
  .then( response => {
    if(response.status == 200) {
      return response.text();
```

por supuesto completamente "fake".

En el primer "then" comprobamos la respuesta del servicio web. Las solicitudes GET para recuperar datos suelen devolver el código 200 cuando todo ha ido bien. Si es así, entonces obtenemos el texto de la respuesta. En caso contrario escalamos un error, que se tratará más adelante en el catch. Este es un buen truco para dejar un único tratamiento de errores para todas las situaciones que se puedan producir.

Otro de los detalles importantes que tendremos que resolver es la conversión del texto de la respuesta, que normalmente será un JSON, para parsearlo y convertirlo en un array con JSON.parse. Una vez tienes el objeto de usuarios harás lo que necesite tu aplicación, ahí ya de pendende de ti.

Conclusión

Miguel Angel Alvarez

Fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Com...

Follow @midesweb 28.8K followers

Manual

¿Alguna duda?

Pregunta y ayuda en la comunidad con tus respuestas en la sección de FAQ

HACER UNA PREGUNTA