El Fetch API en Javascript

https://cursos.mejorcodigo.net/article/el-fetch-api-en-javascript-43

Introducción

La peor experiencia al momento de trabajar con el API común de AJAX en Javascript, XMLHttpRequest, es que no es del todo fácil de utilizar. Existe una mejor alernativa que nos brindan los navegadores Chrome y Firefox para trabajar con este tipo de peticiones. Esta alternativa se llama "**Fetch API**" — en este artículo veremos como implementar este API así como ver algunos aspectos importantes de el.

XMLHttpRequest

Desde nuestro punto de vista, XHR es un bastante complicado. Actualmente, así se utiliza este API:

```
// XHR
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
  peticion = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
  try {
    peticion = new ActiveXObject('Msxml2.XMLHTTP');
  }
  catch (e) {
    try {
      peticion = new ActiveXObject('Microsoft.XMLHTTP');
    }
    catch (e) {}
  }
}

// Abrir y enviar.
peticion.open('GET', 'https://dominio.com/archivo-ajax', true);
peticion.send(null);
```

Lo que puedes observar en el ejemplo anterior es un ejemplo básico utilizando el API de XHR. Creo que la mayoría de nosotros podemos decir que este código es bastante largo y un poco revuelto.

Ejemplo básico utilizando fetch

La función fetch existe dentro de la global window y el primer argumento de la función es la URL.

```
// url (requerida), opciones (opcional)
fetch('https://dominio.com/archivo-ajax', {
  method: 'get'
}).then(function(respuesta) {
  console.log(respuesta)
}).catch(function(err) {
```

```
  // Error :(
});
```

Como podemos ver, este API hace uso de promesas para el manejo de callbacks y resultados.

```
// Ejemplo de manejo de resultados
fetch('https://dominio.com/archivo-ajax', {
  method: 'get'
}).then(function(respuesta) {
  console.log(respuesta)
}).catch(function(err) {
  // Error :(
});

// Podemos encadenar promesas, para un manejo más "avanzado"
fetch('https://dominio.com/archivo-ajax')
.then(function(respuesta) {
  return //...
}).then(function(valorRetornado) {
  // ...
}).catch(function(err) {
  // Error :(
});
```

Si no estas acostumbrado a utilizar el then, tendrás que leer sobre el ya que se utiliza en muchos lados.

Encabezado de Peticiones

La posibilidad de modificar el encabezado de las peticiones es un aspecto muy importante. Puedes modificar el encabezado de cada petición utilizando new Headers():

```
// Crear una instacia vacia de los encabezados
var encabezado = new Headers();

// Agregamos algunos encabezados
encabezado.append('Content-Type', 'text/plain');
encabezado.append('X-Mi-Encabezado', 'ValorAqui');

// Verificar, obtener, establecer valores en el encabezado
encabezado.has('Content-Type'); // true, retorna true si el encabezado existe
encabezado.get('Content-Type'); // "text/plain"
encabezado.set('Content-Type', 'application/json');

// Eliminar un encabezado
encabezado.delete('X-Mi-Encabezado');

// Establecer valores por default
var encabezado = new Headers({
  'Content-Type': 'text/plain',
  'X-Mi-Encabezado': 'ValorAqui'
```

```
});
```

Request (Petición)

Una instancia de Request representa la petición (el envio) de la función fetch. Si le indicamos el Request al fetch podremos enviar peticiones más especificas.

Veamos un ejemplo utilizando Request:

```
var peticion = new Request(
  'https://dominio.com/archivo-ajax',
  {
    method: 'POST',
    mode: 'cors',
    redirect: 'follow',
    headers: new Headers({
      'Content-Type': 'text/plain'
    })
  }
);

// Ahora lo utilizamos
fetch(peticion).then(function() { /* manejo de la respuesta */ });
```

Parámetros de Request

Podemos personalizar aún más las peticiones utilizando lo siguiente:

- method - GET, POST, PUT, DELETE, HEAD
- url - URL de la petición
- headers - el objeto de los encabezados
- referrer - remitente de la solicitud
- mode - cors, no-cors, same-origin
- credentials - ¿enviar cookies con la peticióńnt? omit, same-origin
- redirect - follow, error, manual
- integrity - valor de la integridad
- cache - modo cache (default, reload, no-cache)

Reponse (Respuesta)

El Response (respuesta) representa la parte then de la función fetch. Dentro de esta instancia podemos establecer la estructura del objeto retornado de cada petición.

Veamos un ejemplo:

```
// Creamos la estructura de la respuesta
var respuesta = new Response('.....', {
  ok: false,
  status: 404,
  url: '/'
```

```
});

fetch('https://dominio.com/archivo-ajax')
.then(function(respuesta) {
  console.log('Estado: ', respuesta.status);
  // Hacemos uso del status establecido en el Response
});
```

También podemos personalizar la respuesta utilizando:

- clone() - Crea un clon del objeto Response.
- error() - Retorna un nuevo objeto Response asociado con un error de red.
- redirect() - Crea una nueva respuesta con una URL diferente.
- arrayBuffer() - Retorna una promesa que soluciona un ArrayBuffer.
- blob() - Retorna una promesa que soluciona un Blob.
- formData() - Retorna una promesa que soluciona un objeto FormData.
- json() - Retorna una promesa que soluciona un objeto JSON.

Manejo de objetos JSON

Digamos que quieres enviar una petición y recibir un objeto JSON. La respuesta en el callback contiene una función .json() que nos permite convertir información en texto plano a un objeto de JSON.

```
fetch('https://dominio.com/archivo-ajax')
.then(function(respuesta) {
  // Convertir a JSON
  return respuesta.json();
}).then(function(j) {
  // Ahora 'j' es un objeto JSON
  console.log(j);
});
```

Envio de información de un formulario

Un uso común de AJAX es el envio de información de una etiqueta <form>.

```
fetch('https://dominio.com/enviar-formulario', {
  method: 'post',
  body: new FormData(document.getElementById('formulario-contacto'))
});
```

Utilizando Fetch

https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch

Introducción

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas. También provee un método global fetch() que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.

Este tipo de funcionalidad se conseguía previamente haciendo uso de XMLHttpRequest. Fetch proporciona una mejor alternativa, y también aporta un único lugar lógico en el que definir otros conceptos relacionados con HTTP como CORS y extensiones para HTTP.

La especificación fetch difiere de JQuery.ajax() en dos formas principales:

- El objeto Promise devuelto desde fetch() no será rechazado con un estado de error HTTP incluso si la respuesta es un error HTTP 404 o 500. En cambio, este se resolverá normalmente (con un estado ok configurado a false), y este solo sera rechazado ante un fallo de red o si algo impidio completar la solicitud.
- Por defecto, fetch no enviará ni recibirá cookies del servidor, resultando en peticiones no autenticadas si el sitio permite mantentener una sesión de usuario (para mandar cookies, credentials de la opción init deberan ser configuradas). Desde el 25 de agosto de 2017. La especificación cambió la politica por defecto de las credenciales a same-origin. Firefox cambió desde la versión 61.0b13.

Una petición básica de fetch es realmente simple de realizar. Eche un vistazo al siguente código:

```
fetch('http://example.com/movies.json')
  .then(function(response) {
    return response.json();
  })
  .then(function(myJson) {
    console.log(myJson);
  });
```

Aquí estamos recuperando un archivo JSON a través de red e imprimiéndolo en la consola. El uso de fetch() más simple toma un argumento (la ruta del recurso que quieres obtener) y devuelve un objeto Promise conteniendo la respuesta, un objeto Response.

Esto es, por supuesto, una respuesta HTTP no el archivo JSON. Para extraer el contenido en el cuerpo del JSON desde la respuesta, usamos el método json() (definido en el mixin de Body, el cual está implementado por los objetos Request y Response).

Opciones de petición

El método fetch() puede aceptar un segundo parámetro opcional, un objeto init que permite controlar algunos ajustes:

Vea fetch(), para ver todas las opciones disponibles y más detalles.

```
var misCabeceras = new Headers();

var miInit = { method: 'GET',
               headers: misCabeceras,
               mode: 'cors',
               cache: 'default' };

fetch('flores.jpg',miInit)
.then(function(response) {
  return response.blob();
})
.then(function(miBlob) {
  var objectURL = URL.createObjectURL(miBlob);
  miImagen.src = objectURL;
});
```

Comprobando que la petición es satisfactoria

Una petición promise fetch() será rechazada con TypeError cuando se encuentre un error de red, aunque esto normalmente significa problemas de permisos o similares — por ejemplo, un 404 no constituye un error de red. Una forma precisa de comprobar que la petición fetch() es satisfactoria pasa por comprobar si la promesa ha sido resuelta, además de comprobar que la propiedad Response.ok tiene el valor true. El código sería algo así:

```
fetch('flores.jpg').then(function(response) {
  if(response.ok) {
    response.blob().then(function(miBlob) {
      var objectURL = URL.createObjectURL(miBlob);
      miImagen.src = objectURL;
    });
  } else {
    console.log('Respuesta de red OK.');
  }
})
.catch(function(error) {
  console.log('Hubo un problema con la petición Fetch:' + error.message);
});
```

Proporcionando tu propio objeto Request

En lugar de pasar la ruta al recurso que deseas solicitar a la llamada del método fetch(), puedes crear un objeto de petición utilizando el constructor Request(), y pasarlo como un argumento del método fetch():

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request('flowers.jpg', myInit);
```

```
fetch(myRequest)
.then(function(response) {
  return response.blob();
})
.then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

Request() acepta exactamente los mismos parámetros que el método fetch(). Puedes incluso pasar un objeto de petición existente para crear una copia del mismo:

```
var anotherRequest = new Request(myRequest, myInit);
```

Esto es muy útil ya que el cuerpo de las solicitudes y respuestas son de un sólo uso. Haciendo una copia como esta te permite utilizar la petición/respuesta de nuevo, y al mismo tiempo, si lo deseas, modificar las opciones de init. La copia debe estar hecha antes de la lectura del <body>, y leyendo el <body> en la copia, se marcará como leido en la petición original.

Existe también un método clone() que crea una copia. Este tiene una semántica ligeramente distinta al otro método de copia — el primero fallará si el cuerpo de la petición anterior ya ha sido leído (lo mismo para copiar una respuesta), mientras que clone() no.

Enviar una petición con credenciales incluido

Para producir que los navegadores envien una petición con las credenciales incluidas, incluso para una llamada de origen cruzado, añadimos credentials: 'include' en el el objeto init que se pasa al método fetch().

```
fetch('https://example.com', {
  credentials: 'include'
})
```

Si solo quieres enviar la credenciales si la URL de la petición está en el mismo origen desde donde se llamada el script, añade credentials: 'same-origin'.

```
// El script fué llamado desde el origen 'https://example.com'
fetch('https://example.com', {
  credentials: 'same-origin'
})
```

Sin embargo para asegurarte que el navegador no incluye las credenciales en la petición, usa credentials: 'omit'.

```
fetch('https://example.com', {
  credentials: 'omit'
})
```

Enviando datos JSON

Usa fetch() para enviar una petición POST con datos codificados en JSON .

```
var url = 'https://example.com/profile';
var data = {username: 'example'};

fetch(url, {
  method: 'POST', // or 'PUT'
```

```
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers:{
    'Content-Type': 'application/json'
  }
}).then(res => res.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', response));
```

Enviando un archivo

Los archivos pueden ser subido mediante el HTML de un elemento input <input type="file" />, FormData() y fetch().

```
var formData = new FormData();
var fileField = document.querySelector("input[type='file']");

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', response));
```

Cabeceras

La interfaz Headers te permite crear tus propios objetos de headers mediante el constructor Headers(). Un objeto headers es un simple multi-mapa de nombres y valores:

```
var content = "Hello World";
var myHeaders = new Headers();
myHeaders.append("Content-Type", "text/plain");
myHeaders.append("Content-Length", content.length.toString());
myHeaders.append("X-Custom-Header", "ProcessThisImmediately");
```

Lo mismo se puede lograr pasando un array o un objeto literal al constructor:

```
myHeaders = new Headers({
  "Content-Type": "text/plain",
  "Content-Length": content.length.toString(),
  "X-Custom-Header": "ProcessThisImmediately",
});
```

Los contenidos pueden ser consultados o recuperados:

```
console.log(myHeaders.has("Content-Type")); // true
console.log(myHeaders.has("Set-Cookie")); // false
myHeaders.set("Content-Type", "text/html");
myHeaders.append("X-Custom-Header", "AnotherValue");

console.log(myHeaders.get("Content-Length")); // 11
console.log(myHeaders.getAll("X-Custom-Header")); // ["ProcessThisImmediately",
"AnotherValue"]
```

```
myHeaders.delete("X-Custom-Header");
console.log(myHeaders.getAll("X-Custom-Header")); // [ ]
```

Todos los métodosde de headers lanzan un TypeError si un nombre de cabecera no es un nombre de cabecera HTTP válido. Las operaciones de mutación lanzarán un TypeError si hay un guarda inmutable (ver más abajo). Si no, fallan silenciosamente. Por ejemplo:

```
var myResponse = Response.error();
try {
  myResponse.headers.set("Origin", "http://mybank.com");
} catch(e) {
  console.log("Cannot pretend to be a bank!");
}
```

Un buen caso de uso para headers es comprobar cuando el tipo de contenido es correcto antes de que se procese:

```
fetch(myRequest).then(function(response) {
  var contentType = response.headers.get("content-type");
  if(contentType && contentType.indexOf("application/json") !== -1) {
    return response.json().then(function(json) {
      // process your JSON further
    });
  } else {
    console.log("Oops, we haven't got JSON!");
  }
});
```

Objetos Response

Cómo has visto anteriormente, las instancias de Response son devueltas cuando fetch() es resuelto.

Las propiedades de response que usarás son:

- **Response.status** — Entero (por defecto con valor 200) que contiene el código de estado de las respuesta.
- **Response.statusText** — Cadena (con valor por defecto "OK"), el cual corresponde al mensaje del estado de código HTTP.
- **Response.ok** — Visto en uso anteriormente, es una clave para comporbar que el estado está dentro del rango 200-299 incluido. Este devuelve un valor Boolean.

Estos pueden tambíen creados programaticamente a través de JavaScript, pero esto solo es realmete util en ServiceWorkers, cuando pones un objeto response personalizado a una respuesta recivida usando un método respondWith():

```
var myBody = new Blob();

addEventListener('fetch', function(event) {
  event.respondWith(
    new Response(myBody, {
      headers: { "Content-Type" : "text/plain" }
    })
  );
});
```

El constructor Response() toma dos argurmentos opcionales, un cuerpo para la respuesta y un objeto init (similar al que acepta Request()).

Body

Tanto las peticiones como las respuestas pueden contener datos body. Body es una instancia de cualquiera de los siguientes tipos:

- ArrayBuffer
- ArrayBufferView (Uint8Array y amigos)
- Blob/File
- string
- URLSearchParams
- FormData

El mixin de Body define los siguientes metodos para extraer un body (implementado por {domxref("Request")}} and Response). Todas ellas devuelven una promesa que es eventualmente resuelta con el contenido actual.

- arrayBuffer()
- blob()
- json()
- text()
- formData()

Este hace uso de los datos no texttuales mucho mas facil que si fuera con XHR.

Las peticiones body pueden ser establecidas pasando el parametro body:

```
var form = new FormData(document.getElementById('login-form'));
fetch("/login", {
  method: "POST",
  body: form
});
```

Tanto peticiones y respuestas (y por extensión la function fetch()), intentaran inteligentemente determinar el tipo de contenido. Una petición tambien establecerá automáticamente la propiedad Context-Type de la cabecera si no es ha establecido una.

Detectar característica

Puedes comprobar si el navegador soporta la API de Fetch comprobando la existencia de Headers, Request, Response o fetch() sobre el ámbito de Window o Worker. Por ejemplo:

```
if (self.fetch) {
    // run my fetch request here
} else {
    // do something with XMLHttpRequest?
}
```

Polyfill

Para utilizar fetch() en un explorador no soportado, hay disponible un Fetch Polyfill que recrea la funcionalidad para navegadores no soportados.