Peticiones HTTP

AJAX : Asynchronous JavaScript And XML, es una técnica de desarrollo web para crear aplicaciones interactivas.

- Protocolos de Transferencia de Hipertexto
- Objeto XMLHttpRequest

  Comprobar qué navegadores soportan el objeto XMLHttpRequest:

  - ◊ https://caniuse.com

- Estados HTTP

Sumario

Objeto XMLHttpRequest

XMLHttpRequest es un objeto JavaScript que fue diseñado por Microsoft y adoptado por Mozilla, Apple y Google. Actualmente es un estándar de la W3C. Proporciona una forma fácil de obtener información de una URL sin tener que recargar la página completa. Una página web puede actualizar sólo una parte de la página sin interrumpir lo que el usuario está haciendo. XMLHttpRequest es ampliamente usado en la programación AJAX.

A pesar de su nombre, XMLHttpRequest puede ser usado para recibir cualquier tipo de dato, no solo XML, y admite otros formatos además de HTTP (incluyendo file y ftp).

Ejemplo 1.- Leer un archivo .txt por GET

En este ejemplo vamos a cargar el contenido de un archivo .txt de modo dinámico y, su contenido, lo cargaremos a la web.

El contenido del archivo 'info.txt' que vamos a cargar será una lista:

- **TXT**

```
<ol>
<li>Frutas:
<ul>
<li>Manzana</li>
<li>Naranja</li>
<li>Melón</li>
</ul>
</li>
<li>Legumbres</li>
<li>Cereales</li>
</ol>
```

- **HTML:**

```
<!DOCTYPE html>
  <html lang="es">
    <head>
      <meta charset="utf-8" />
      <title>Primer ejemplo AJAX</title>
    </head>
```

```
    <body>
      <form name="formulario">
        <p>
          <input type="button" value="Leer archivo" id="lee" />
        </p>
      </form>
      <div id="etiqueta"></div>
      <script src="script.js"></script>
    </body>
  </html>
```

- **JavaScript:**

```
function obtenerInfo() {
  let xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {

//Indicamos qué queremos que realice la web antes de enviar la petición al servidor
    if (this.readyState == 4 && this.status == 200) {   //Visitar developer.Mozilla para comprobar propiedades 'readyState' y 'status
      document.getElementById("etiqueta").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "info.txt", true);
  xhttp.send();
}
//Añadimos escuchadores
document.getElementById('lee').addEventListener('click', obtenerInfo, false);
```

Ejemplo 2.- Leer un archivo .php por GET

Podemos leer dinámicamente, sin recargar la página, un archivo .php como el siguiente. Este archivo php también podría ser mucho más complejo, por ejemplo un archivo que accediese a una base de datos e interactuase con ella para leer, escribir, borrar registros... etc.

Lo primero será crear el archivo **respuestaSimple.php** con el siguiente contenido.

- **PHP**

```
<?php
  header("Content-Type: text/html; charset=utf-8");
  echo (
   "<ol>
   <li>Frutas:
   <ul>
   <li>Manzana</li>
   <li>Naranja</li>
   <li>Melón</li>
   </ul>
   </li>
   <li>Legumbres</li>
   <li>Cereales</li>
   </ol>"
  );
?>
```

Y modificamos el contenido del archivo **scripts.js** del siguiente modo. Aprovechando para configurarlo de modo que evite la caché del navegador.

- **JavaScript**

```
function obtenerInfo() {
  let xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("etiqueta").innerHTML = this.responseText;
    }
  };
  let url = "info.txt?r=" + (new Date()).getTime();
  xhr.open("GET", url, true);
  xhr.send();
}
```

```
//Añadimos escuchadores
document.getElementById('lee').addEventListener('click', obtenerInfo, false);
```

Ejemplo 3: Acceso a un JSON por GET

Utilizar la tecnología AJAX con el objeto **XMLHttpRequest** sólo es posible haciendo peticiones al propio servidor. A no ser, que el servidor web esté configurado para habilitar el uso compartido de recursos (CORS).

- Habilitar CORS en Apache2
- Habilitar CORS EN Nginx

En este tercer ejemplo, accedemos a un JSON de la web externa JSON Placeholder, que nos permite realizar peticiones AJAX pues tiene habilitado el uso compartido de recursos (CORS).

Aprovecharemos para realizar de un modo más preciso las peticiones AJAX con el objeto **XMLHttpRequest**.

- **HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX</title>
</head>
<body>
    <h1 id='titulo'>AJAX</h1>
    <button id='boton'>Pide datos</button>


    <script src="scripts.js"></script>
</body>
</html>
```

- **JavaScript**

```
function leerDatos() {
    //Crear el xhr haciendo que funcione en navegadores viejos
    let xhr;
    if (window.XMLHttpRequest) xhr = new XMLHttpRequest();
    else xhr = new ActiveXObject("Microsoft.XMLHTTP");

    //Antes de enviar la petición le decimos lo que queremos que haga con los datos
    ///El siguiente código, como mucho va delante del envío de la petición .send()
    ///Evento 'load' = La petición ya terminó
    xhr.addEventListener('load', function(datos) {
        //La parte que nos va a interesar es 'target'
        ///Y, dentro de 'target' lo que nos interesa es 'response' (todos los datos)
    //console.log(datos.target.response);
        ///Vemos el 'typeof' de esos datos y nos devuelve 'string' NO un objeto
    //console.log(typeof datos.target.response);
        ///Convertimos ese 'string' en un 'objeto'
        const datosJSON = JSON.parse(datos.target.response)
        console.log(dataJSON);

    })

    // Seleccionamos método 'GET'
    const url = 'https://jsonplaceholder.typicode.com/users';
    xhr.open('GET', url)

    //Enviamos la petición
    xhr.send();

}
```

Ahora, modificamos los códigos para mostrar por pantalla el 'id' y el 'name' de cada uno de los usuarios descargados. Para ello crearemos una lista desordenada:

- **HTML**

```
// ...

    <ul id="lista"></ul>

// ...
```

- **JavaScript**

```
//...
        const lista = document.getElementById('lista');
        for (const userInfo of datosJSON) {
            const listaI = document.createElement('li');
            listaI.textContent = `${userInfo.id} -> ${userInfo.name}`;
            lista.appendChild(listaI);
//...
```

Ejemplo 4: Formulario con GET y POST

En el siguiente ejemplo realizamos el autocompletado de un formulario empleando GET y el envío por POST

- **HTML**

```
<head>
<meta charset="utf-8" />
<title>Ejemplo AJAX GET + POST</title>
</head>

<body>
<h1>Pide Pizza</h1>


<form name="formulario" method="post" action="ejemploPost.php">
<p>Introduzca su n&uacute;mero:
<input type="text" size="14" name="telefono" id="telefono" />
</p>
<p>Su pizza ser&aacute; entregada en la direcci&oacute;n:</p>
<div id="direccion"></div>
<p>Escriba su solicitud aqu&iacute;:</p>
<p>
<textarea name="solicitud" rows="6" cols="50" id="solicitud"></textarea>
</p>
<p>
<input type="submit" value="Pedir una Pizza" id="submit" />
</p>
</form>

<script src="scripts.js"></script>

</body>
</html>
```

- **PHP**

```
//buscarCliente.php
<?php
if ($_GET['phone']=="123") {
echo ("Pizza Tropical|Rúa Fray Rosendo Salvado Nº 25, 3º C\n15705 Santiago de Compostela\nA Coruña");
} elseif ($_GET['phone']=="1234") {
echo ("Pizza Margarita|Rúa do Sar Nº 40, 1º C\n15702 Santiago de Compostela\nA Coruña");
} else {
echo "No esta en la base de datos.|No existe.";
}
?>

//ejemploPost.php
<?php

$telefono = $_POST['telefono'];
```

```
$solicitud = $_POST['solicitud'];

echo "<p>Tu teléfono es: ".$telefono."</p>";
echo "<p>Tu solicitud es: ".$solicitud."</p>";

?>
```

- **JavaScript**

```
//Utilizamos el objeto AJAX creado para pedir información asíncrona al servidor
function obtenerInfoCliente() {

    //Creamos el objeto XMLHttpRequest()
    let xhr;
    if (window.XMLHttpRequest) xhr = new XMLHttpRequest();
    else xhr = new ActiveXObject("Microsoft.XMLHTTP");


    let phone = document.getElementById("telefono").value;
    let url = "buscarCliente.php?phone=" + encodeURIComponent(phone) + "&r=" + (new Date()).getTime();
     console.log(url);

    //Evento 'load' = La petición ya terminó
    xhr.addEventListener('load', function(datos) {
        //Lo que nos interesa es : datos.target.response
        const respuesta = datos.target.response;
        // console.log(respuesta);

        const aRespuesta = respuesta.split("|"); // Hacemos un array
        console.log(aRespuesta);

        // Seleccionamos el textarea con id 'solicitud' y lo rellenamos con tu último pedido
        document.getElementById("solicitud").value = aRespuesta[0];

        // Seleccionamos el div con id 'direccion' y mostramos la dirección si ya la tenemos
        if (aRespuesta[1] != undefined)
            document.getElementById("direccion").innerHTML = aRespuesta[1].replace(/\n/g, "<br>");
    });

    xhr.open("GET", url, true);
      //Método GET
      //URL con el parámetro que nos interese
      //true = Asíncrono

     xhr.send();
      //Se envía la petición, vacía por ser get
}
//-----------------------------------------------------------------------------

//Añadimos escuchador en la caja de texto con id 'telefono'
///'change' es cuando pierde el foco
document.getElementById('telefono').addEventListener('change', obtenerInfoCliente, false);
```

Ejemplos más elaborados

- Práctica 1 XMLHttpRequest ? GET + PHP ? MySQL
- ...

Objeto Fetch

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales 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 alternativa mejor. Fetch también aporta un único lugar lógico en el que definir otros conceptos relacionados con HTTP como CORS y extensiones para HTTP.

El **fetch()** devolverá una Promesa que será aceptada cuando reciba una respuesta y sólo será rechazada si hay un fallo de red o si por alguna razón no se pudo completar la petición. El modo más habitual de manejar las promesas es utilizando **.then()** tal y como veremos en los ejemplos siguientes.

Al método **.then()** se le pasa una función **callback** donde su parámetro **res** (respuesta) es el objeto de respuesta de la petición que hemos realizado. En su interior realizaremos la lógica que queramos hacer con la respuesta a nuestra petición. A la función **fetch(url, options)** se le pasa por parámetro la url de la petición y, de forma opcional, un objeto **options** con opciones de la petición HTTP:

- method: GET,HEAD,POST,PUT
- body
- headers
- credentials

Más detalles en esta web.

Veamos unos ejemplos.

GET con Fetch

Realizaremos ahora el ejemplo 3 del apartado **XMLHttpRequest()** pero utilizando **fetch()**.

- **HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX</title>
</head>
<body>
    <h1 id='titulo'>AJAX</h1>
    <button id='boton'>Pide datos</button>

    <br>
    <ul id="lista"></ul>


    <script src="scripts.js"></script>
</body>
</html>
```

- **JavaScript**

```
const boton = document.getElementById('boton');
boton.addEventListener('click', () => {

    const url = 'https://jsonplaceholder.typicode.com/users';
    fetch(url)
        //Elementos recibidos son un json,
        ///con método .json() los convertimos a un objeto javascript json
        .then(res => res.json())
        //'res' ahora ya es un json y lo podemos recorrer con 'for of'
        .then(res => {
            const lista = document.getElementById('lista');
            //Borramos lista existente
            lista.innerHTML = "";
            //Creamos un fragmento de código con todos los elementos de la lista
            const fragmento = document.createDocumentFragment();
            for (const userInfo of res) {
                const li = document.createElement('li');
                li.textContent = `${userInfo.id} - ${userInfo.name}`;
                fragmento.appendChild(li);
            }
            //Añadimos el fragmento con los li al ul
            lista.appendChild(fragmento);
        })
        .catch(err => {
            console.log('Error encontrado: ' + err.message);
        });
});
```

Librería Axios

Se trata de una librería Javascript capaz de ejecutarse tanto en el navegador como en NodeJS, que facilita todo tipo de operaciones como cliente HTTP.

Con Axios puedes realizar solicitudes contra un servidor, completamente configurables, y recibir la respuesta de una manera sencilla de procesar. En este artículo te explicaremos las ventajas de esta librería, así como algunos ejemplos básicos de uso, con los que podrás observar su funcionamiento.

La librería está basada en promesas, por lo que al usarla podremos generar un código asíncrono bastante ordenado. Incluso es capaz de usar **Async** / **Await** para mejorar la sintaxis.

Volver