Desarrollo Web en Entorno Cliente

UD 08. Introducción a AJAX

Actualizado Noviembre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

ÍNDICE DE CONTENIDO

UD08. INTRODUCCIÓN A AJAX

1. ¿QUÉ ES AJAX?

En primer lugar, aclarar en nuestro contexto informático que no es AJAX:

- Un equipo de fútbol holandés.
- Productos de limpieza.

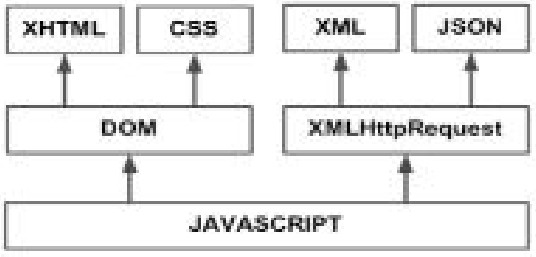
En nuestro caso, AJAX es el acrónimo de "Asynchronous Javascript And XML" (Javascript asíncrono y XML). AJAX en si no es una tecnología, sino un conjunto de tecnologías.

Tenéis más información en https://es.wikipedia.org/wiki/AJAX

2. TECNOLOGÍAS PRESENTES EN AJAX

Las tecnologías presentes en AJAX son:

- XHTML y CSS para la presentación de la página.
- DOM para la manipulación dinámica de elementos de la página.
- Formatos de intercambio de información como JSON o XML.
- El objeto XMLHttpRequest, para el intercambio asíncrono de información (es decir, sin recargar la página).
- Javascript, para aplicar las anteriores tecnologías.



La forma de trabajar es la siguiente: Javascript se encarga de unir todas las tecnologías. Para manipular la parte de representación de la página utiliza DOM (así manipula el XHTML y el CSS).

Para realizar peticiones asíncronas usa el objeto XMLHttpRequest. Este objeto intercambia información que son simplemente cadenas de texto. Cuando se quieren formatear objetos más complejos, se suele utilizar JSON o XML.

Durante el curso utilizaremos habitualmente JSON para intercambiar la información.

3. FUNCIONAMIENTO DE APLICACIÓN WEB CLÁSICA VS APLICACIÓN WEB AJAX

En una aplicación Web clásica:

1. El cliente hace una petición al servidor.
2. El servidor recibe la petición.
3. El servidor procesa la petición y genera una nueva página con la petición procesada. (Ejemplo, se añade un post a un foro).
4. El cliente recibe la nueva página completa y la muestra.

En una aplicación Web AJAX:

1. El cliente hace una petición asíncrona al servidor.
2. El servidor recibe la petición.
3. El servidor procesa la petición y responde asíncronamente al cliente.
4. El cliente recibe la respuesta y con ella modifica dinámicamente los elementos afectados de la página sin recargar completamente la página.

Las aplicaciones Web AJAX son mejores ya que reducen la cantidad de información a intercambiar (no se envía la página entera, sino que se modifica solo lo que interesa) y a su vez al usuario final le da una imagen de mayor dinamismo, viendo una página web como una aplicación de escritorio.

Visitando el agregador de noticias https://www.meneame.net/ y "meneando" cualquier noticia, podéis ver como funciona AJAX (el contador de "meneos" aumenta, pero la página no se ha recargado).

4. OBJETO XMLHTTPREQUEST

En esta unidad didáctica hablaremos de cómo funciona el objeto **XMLHttpRequest**, imprescindible para utilizar AJAX desde Javascript. Este objeto nos permitirá hacer peticiones asíncronas y se encargará de avisarnos cuando se reciba su respuesta.

5. PRINCIPALES MÉTODOS DEL OBJETO XMLHTTPREQUEST

En https://es.wikipedia.org/wiki/XMLHttpRequest se puede observar una definición del objeto y sus principales atributos y métodos.

Tomado de Wikipedia:

5.1 Atributos

Atributo	Descripción
readyState	Devuelve el estado del objeto como sigue: ● 0 = sin inicializar ● 1 = abierto ● 2 = cabeceras recibidas ● 3 = cargando ● 4 = completado.
responseBody	Devuelve la respuesta como un array de bytes.
responseText	Devuelve la respuesta como una cadena.
responseXML	Devuelve la respuesta como XML. Esta propiedad devuelve un objeto documento XML, que puede ser examinado usando las propiedades y métodos del árbol del Document Object Model.
status	Devuelve el estado como un número (p. ej. 404 para "Not Found" y 200 para "OK").
statusText	Devuelve el estado como una cadena (p. ej. "Not Found" o "OK").

5.2 Métodos

Método	Descripción
abort()	Cancela la petición en curso.
getAllResponseHeaders()	Devuelve el conjunto de cabeceras HTTP como una cadena.
getResponseHeader (nombreCabecera)	Devuelve el valor de la cabecera HTTP especificada.
open (método, URL, [asíncrono [nombreUsuario [clave]]])	Especifica el método, URL y otros atributos opcionales de una petición. El parámetro de método puede tomar los valores "GET", "POST", o "PUT" ("GET" y "POST" son dos formas para solicitar datos, con "GET" los parámetros de la petición se codifican en la URL y con "POST" en las cabeceras de HTTP). El parámetro URL puede ser una URL relativa o completa. El parámetro asíncrono especifica si la petición será gestionada asíncronamente o no. Un valor true indica que el proceso del script

	continúa después del método send(), sin esperar a la respuesta, y false indica que el script se detiene hasta que se complete la operación, tras lo cual se reanuda la ejecución. En el caso asíncrono se especifican manejadores de eventos, que se ejecutan ante cada cambio de estado y permiten tratar los resultados de la consulta una vez que se reciben, o bien gestionar eventuales errores.
send([datos])	Envía la petición.
setRequestHeader (etiqueta, valor)	Añade un par etiqueta/valor a la cabecera HTTP a enviar.

5.3 Eventos

Propiedad	**Descripción**
onreadystatechange	Evento que se dispara con cada cambio de estado.
onabort	Evento que se dispara al abortar la operación.
onload	Evento que se dispara al completar la carga.
onloadstart	Evento que se dispara al iniciar la carga.
onprogress	Evento que se dispara periódicamente con información de estado.

6. FORMA MÁS COMÚN DE UTILIZAR XMLHTTPREQUEST

6.1 Instanciando el objeto

En primer lugar, indicar que debemos inicializar el objeto.

```
httpRequest = new XMLHttpRequest();
```

Esto es válido para la mayoría de navegadores actuales.

Si queremos compatibilidad con navegadores antiguos que soporte ActiveX (Tipo Internet Explorer 6) se puede hacer una función más compleja para obtener el objeto.

```
function obtainXMLHttpRequest()
{
      let httpRequest;
      if (window.XMLHttpRequest){
            //El explorador implementa la interfaz de forma nativa
```

```
            httpRequest = new XMLHttpRequest();
      }
      else if (window.ActiveXObject){
            //El explorador permite crear objetos ActiveX
            try {
                  httpRequest = new ActiveXObject("MSXML2.XMLHTTP");
            } catch (e) {
                  try {
                        httpRequest = new
ActiveXObject("Microsoft.XMLHTTP");
                  } catch (e) {}
            }
      }
      // Si no se puede crear, devolvemos false. En caso contrario,
devolvemos el  objeto
      if (!httpRequest){
            return false;
      }
      else{
            return httpRequest;
      }
}
```

6.2 Comportamiento evento onreadystatechange

Tras ello, decidiremos el comportamiento del evento "onreadystatechange", evento que se producirá cada vez que haya producido un cambio en el atributo "ready".

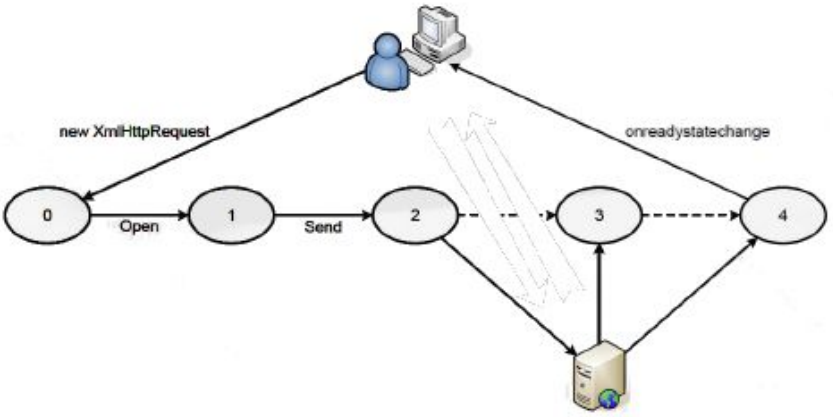
Antes de hacer nada, deberemos explicar 3 métodos del objeto:

- open(Metodo,URL,true). Este método recibe qué método de comunicación utiliza en la petición (GET o POST) y que URL se aplica.
- send([datos]). Este método ha de hacerse posteriormente a un open, nunca antes. Envía la información a la URL especificada en open.
- setRequestHeader que indicará el formato de las cabeceras enviadas.

También debemos explicar el atributo readyState: este atributo puede poseer los siguientes valores:

- 0 al inicializarse el objeto.
- 1 al abrirse una conexión (al usar el método open).
- 2 al hacer una petición (uso de send).
- 3 mientras se está recibiendo información de la petición.
- 4 cuando la petición se ha completado.

Esta imagen resume el proceso.



¿Cómo lo aplicamos en código? El evento onreadystatechange cada vez que se produzca comprobará en que estado nos encontramos y hará lo planeado para ese estado. Generalmente el estado más utilizado es el 4, donde se ha completado la operación.

Ejemplo: Un ejemplo de todo esto donde se hace una petición y al finalizarse si el estado es correcto (200) se actualiza un div con id=capa en el código.

```
// Abrimos la conexión
httpRequest.open("POST","ajax.php",true);
// Indicamos como seran las cabeceras
httpRequest.setRequestHeader("Content-Type","application/x-www-form-urle
ncoded");
// Definimos el comportamiento de onreadystatechange
httpRequest.onreadystatechange=function(){
      if (httpRequest.readyState==1) {
            document.getElementById('capa').innerHTML="CARGANDO...";
      }
      // Si se ha completado
      if (httpRequest.readyState==4) {
            // Si es correcto el status
            if (httpRequest.status==200){
            // de httpRequest.responseText obtenemos la cadena con la
respuesta
                  document.getElementById('capa').innerHTML=httpRequest
.responseText;
            }
      }
}
// Enviamos la acción
httpRequest.send("accion="+accion);
```

En este ejemplo, abrimos con open, definimos el comportamiento del evento "onreadystatechange" y cuando está todo listo, enviamos la petición con "send".

Para ver un ejemplo completo, consulte los ejemplos completos adjuntos. En los ejemplos se incluirá el código del servidor a modo informativo y para pruebas (para el examen sólo examinaremos del cliente).

7. PROBANDO LOS EJEMPLOS

Para probar los ejemplos, necesitaréis un servidor al que hacer peticiones asíncronas.

Para ejemplos y ejercicios que mandemos en los próximos temas, subiré los ficheros de servidor a mi servidor https://apuntesfpinformatica.es

Los ficheros se encontrarán en https://apuntesfpinformatica.es/DWEC/loquesea.php donde "loquesea.php" será el nombre del fichero que se encuentra alojado en el servidor.

Los servidores prohíben normalmente hacer peticiones AJAX a sitios de fuera de su dominio. Para aceptar esto, se puede añadir en el servidor:

```
header("Access-Control-Allow-Origin: *");
```

Esto nos permitirá que desde otros dominios, le puedan llegar peticiones AJAX.

Los ejemplos subidos a mi servidor incluyen esa línea, así que deberían funcionar correctamente.

Si queréis montar un servidor en local para pruebas, estos video tutoriales pueden seros útiles:

- Instalar LAMP en Ubuntu https://www.youtube.com/watch?v=f-l07vKQLW0
- Instalar WAMP en Windows https://www.youtube.com/watch?v=aUa2-0l2ZXc

8. AJAX MODERNO: USANDO "FETCH" EN LUGAR DEL OBJETO XMLHTTPREQUEST

En las versiones modernas de los motores de Javascript, se ha incluido una nivea forma de realizar peticiones asíncronas basada en promesas y más acordes a estilos modernos de programación que el uso del objeto XMLHttpRequest: el uso de "fetch".

La documentación oficial sobre "fetch" la podéis encontrar en:
https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch

Vamos a mostrar un pequeño ejemplo comentado, enviando el contenido usando "QueryString" y recibiendo el contenido como texto:

```
// Establecemos a que direccion realizar fetch
fetch("https://miurl.com/web.php", {
      // Establecemos método POST
      method:"POST",
      // Generamos queryString
      body:"accion="+parametro,
      // Indicamos en las cabeceras como es el contenido que enviamos
```

```
      headers: {
            'Content-type': 'application/x-www-form-urlencoded'
      }
      // Código a ejecutar al recibir la respuesta
      }).then(function(response){
            // Si la respuesta es correcta
            if (response.ok) {
            // Código para llamar funcion / hacer función anónima,
            // que gestione código de la respuesta
            // en este ejemplo, "respuesta" contiene texto
            //devuelto por el servidor
                  response.text().then( function (respuesta) {
                  document.getElementById('capa').innerHTML=respuesta;
                  });
            }
      });
});
```

Aquí el mismo ejemplo, pero enviando el contenido como JSON y recibiendo contenido como JSON

```
// Establecemos a que direccion realizar fetch
fetch("https://miurl.com/web.php", {
      // Establecemos método POST
      method:"POST",
      // Generamos JSON
      body:JSON.stringify( {
            accion: parametro
      }),
      // Indicamos en las cabeceras como es el contenido que enviamos
      headers: {
            'Content-type': 'application/json; charset=UTF-8'
      }
      // Código a ejecutar al recibir la respuesta
      }).then(function(response){
            // Si la respuesta es correcta
            if (response.ok) {
            // Código para llamar funcion / hacer función anónima,
            // que gestione código de la respuesta
            // en este ejemplo, "respuesta" contiene JSON
```

```
            //devuelto por el servidor
                  response.json().then( function (respuesta) {

document.getElementById('capa').innerHTML=respuesta.accion;
                  });
            }
      });
});
```

Si queréis saber más sobre “fetch”, podéis revisar los siguientes enlace:

- https://desarrolloweb.com/articulos/fetch-ajax-javascript.html
- https://gomakethings.com/how-to-send-data-to-an-api-with-the-vanilla-js-fetch-method/

9. MATERIAL ADICIONAL

[1] Intro to AJAX (Udacity) https://www.udacity.com/course/intro-to-ajax--ud110

10. BIBLIOGRAFÍA

[1] Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

[3] XMLHttpRequest https://es.wikipedia.org/wiki/XMLHttpRequest

11. AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- García Barea, Sergi
- Folgado Galache, Laura
- Ibáñez Català, Xavier