AJAX

Asynchronous JavaScript And XML permite actualizar páginas web de forma asíncrona utilizando objetos XMLHttpRequest (que están construidos, *built-in*, en el navegador) para pedir datos al servidor web y HTML DOM y JavaScript para mostrarlos o utilizarlos.

Sirve para mandar y recibir datos de un servidor web en segundo plano después de que la página se haya cargado, y actualizarla sin recargarla: mediante DOM se modifican los contenidos de forma asíncrona (tras hacer la petición al servidor la página sigue funcionando mientras espera la respuesta, cuando esta llega se maneja).

Aunque lleve XML en el nombre no siempre se utiliza XML para transferir los datos: también se puede utilizar JSON, texto plano (txt) o ficheros de scripts (asp, php) que pueden realizar acciones en el servidor antes de devolver la respuesta.

La página web y el texto que se carga deben estar en el mismo servidor.

CÓMO FUNCIONA

Evento en la página > JS crea un objeto XMLHTTpRequest > el objeto manda una petición al servidor > el servidor procesa la petición > el servidor manda la respuesta a la página > JS lee la respuesta y actúa en consecuencia.

ELEMENTOS BÁSICOS

1. Se crea el objeto XMLHttpRequest: var xhr = new XMLHttpRequest();
2. Se asigna una función al evento onreadystatechange: xhr.onreadystatechange = f(){}
3. Se abre la petición y se envía: xhr.open(…), xhr.send()

El siguiente ejemplo escribe el contenido del fichero en el elemento id.

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
        if(this.readyState == 4 && this.status == 200){
                document.getElementById("id").innerHTML = this.responseText;
          }
}
xhr.open("GET", "fichero.txt", true);
xhr.send();
```

OBJETO XMLHTTPREQUEST

Métodos

abort() cancela la petición actual

getAllResponseHeaders() devuelve toda la información de la cabecera del recurso (length, server-type, content-type, last-modified, etc.)

getResponseHeader() devuelve información específica de la cabecera.
Ej. this.getResponseHeader("Last-Modified");

open(*method, url, async, user, psw*) define la petición (GET/POST, localización del fichero, true/false, optional user name, optional password)

send() envía la petición al servidor (se usa para peticiones GET)

send(*string*) envía la petición al servidor (se usa para peticiones POST)

setRequestHeader() añade una cabecera (clave-valor) a la petición (header name, header value)

Propiedades

onreadystatechange es un manejador de evento que guarda el estado de la petición: cada vez que el servidor devuelve información del estado (*readyState*) se produce el evento; se le asocia la función que se invocará cuando cambie la propiedad readyState.

readyState guarda el estado del objeto (0 no inicializado, 1 conexión con el servidor establecida, 2 petición recibida, 3 petición procesándose, 4 petición finalizada y respuesta preparada)

responseText devuelve los datos de la respuesta como una cadena (si la respuesta viene en forma de texto plano)

responseXML devuelve los datos de la respuesta como un objeto XML DOM (si la respuesta es un XML). El objeto XMLHttpRequest tiene un conversor de XML que se puede utilizar así para obtener los datos:

```
xmlDoc = xhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
  txt += x[i].childNodes[0].nodeValue + "<br>";
  }
document.getElementById("demo").innerHTML = txt;
xhttp.open("GET", "cd_catalog.xml", true);
xhttp.send();
```

status devuelve el código de estado de la petición (200 OK, 403 Forbidden, 404 Not Found, etc)

statusText devuelve el texto del estado (OK, Not Found, etc.)

La respuesta está preparada cuando readyState es igual a 4 y status igual a 200.

MANDAR LA PETICIÓN

Para mandar la petición al servidor se utilizan los métodos:

open(method, url, async) especifica el tipo de petición (GET/POST, fichero, true/false). Generalmente se mandará true, así se pueden ejecutar otros scripts mientras vuelve la respuesta. Las peticiones síncronas se pueden utilizar para pruebas rápidas (en este caso no haría falta el evento onreadystatechange porque se esperará a la respuesta para continuar con la ejecución), aparte de ese caso no son recomendables porque si el servidor está ocupado o lento la aplicación se parará.

send() envía la petición al servidor (se usa para GET)

send(string) envía la petición al servidor (se usa para POST)

El método GET es más rápido que POST y se puede usar en casi todos los casos.

El método POST es necesario cuando no sirve un fichero en caché (para que se actualice el contenido del servidor), cuando se mandan grandes cantidades de datos al servidor (GET tiene límite de tamaño) y cuando se mandan inputs del usuario (que pueden contener caracteres desconocidos: POST es más robusto y seguro).

Para utilizar el método GET y aún así consultar los datos actualizados (no los del caché), se puede añadir al nombre del fichero ?=" + Math.random()

xhttp.open("GET", "fichero.txt?=" + Math.random(), true);

De esta misma forma se puede añadir información para enviar: "fichero.asp?fname=Hannah&lname=Ford"

Para datos POST como un formulario HTML se añade una cabecera HTTP con setRequestHeader(). Hay que especificar los datos que se quieren enviar con el método send().

xhttp.open("POST", "ajax_test.asp", true);

xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http.send("fname=Jane&lname=Fonda");

TRABAJANDO CON JSON

Al recoger la información de un .json, AJAX recoge la información como texto plano así que es necesario parsearla para poder trabajar con la estructura json: **JSON.parse**("cadena de texto");

var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');

VARIAS TAREAS AJAX

Si tenemos varias tareas en una página, deberíamos crear una sola función para ejecutar el objeto XMLHttpRequest y después crear una función callback para cada tarea AJAX. Estas funciones deberían contener una url y el nombre de otra función a la que invocar tras la respuesta.

```
<button onclick="loadDoc("url-1", myFunction1)">Change content</button>
loadDoc("url-2", myFunction2);
function loadDoc(url, cFunction) {
  var xhttp;
  xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);
    }
  };
  xhttp.open("GET", url, true);
  xhttp.send();
}
function myFunction1(xhttp) {   // content is changed here  }
function myFunction2(xhttp) {  // action goes here    }
```