Ajax nivel 2

XMLHttprequest level 2

jnll

Es una mejora del viejo objeto xmlhttprequest

Como novedades podemos resaltar los siguientes puntos:

- Transferencia de binario
- Eventos para controlar la evolución de la petición
- Peticiones cross-site (diferentes servidores)
- Establecer tiempo de respuesta
- Mejra la forma de pasos de datos con el objeto FormData

El constructor del objeto es ***XMLHttprequest()***

```
objajax=XMLHttprequest();
```

Este constructor retorna un objeto que nos permite comenzar una solicitud y escuchar eventos para controlar el progreso y ejecución de la solicitud.

Métodos del objeto XMLHttprequest

- open(método,url,asíncrono). Configura un solicitud pendientes.

  Los parámetros que recibe son:

  - método: Tipo de método http usado en la conexión. Puede ser POST o GET.
  - url: Código al que se conecta. Fichero que contiene el código que procesa la solicitud. (página web,php,xml,json.....)
  - asíncrono: Es un valor boleano para declarar la conexión síncrona (false) o asíncrona (true). Si realizamos una petición síncrona, javascript no continua su ejecución hasta la finalización de la petición.

- send(datos). Este método inicia la solicitud de ajax. El parámetro que recibe (datos) pude ser de diferentes tipos de datos. El parámetro puede omitirse. En esta primera entrega trabajaremos con texto y el objeto FormData. En la siguiente veremos que podemos declara un ArrayBuffer o un Blob.

- overrideMimeType(mime). Reescribe el tipo mime retornado por el servidor. Recordar que el tipo mime indica al navegador que tipo de fichero que recibe.

Existen más métodos que el alumno estudiará en el futuro.

Eventos

Los eventos con los que podemos trabajar son los siguientes:

- loadstart: Este evento es lanzado cuando empieza la solicitud.

- progress: Es un evento que se lanza cada 50 milisegundos aproximadamente, mientras envía o recibe datos.

- abort: Este evento se lanza cuando se aborta la solicitud.

- timeout: Entre las propiedades de la solicitud existe la propiedad *tiemout*. Cuando se indica un valor a esta propiedad, en milisegundos, y se sobrepasa se lanza este evento.

- loadend: Este evento se lanza cuando la solicitud ha sido completada. No se tiene en cuenta si la solicitud tuvo éxito o no.

- load: Se lanza cuando la solicitud se ha completado con éxito.

- readystatechange: Es un evento que se lanza cuando cambia el valor de la propiedad *readyState*.
  - readyState. Es un propiedad que toma los siguientes valores:
    - 1 → opened – El método open() ha sido llamado
    - 2 → headers-received – El método send() ha sido llamado y las cabeceras y los estados están disponibles.
    - 3 → loading – bajando, responseText contiene datos parciales
    - 4 → done – La operación a terminado.

Propiedades “response”

Esta propiedad recoge los datos devueltos desde el servidor. Existen tres tipos de propiedades response.

- response: Propiedad de propósito general. Retorna la respuesta de la solicitud según el atributo *responseType*, que estudiaremos más adelante.

- responseText: Propiedad que retorna la respuesta de la solicitud en formato texto.

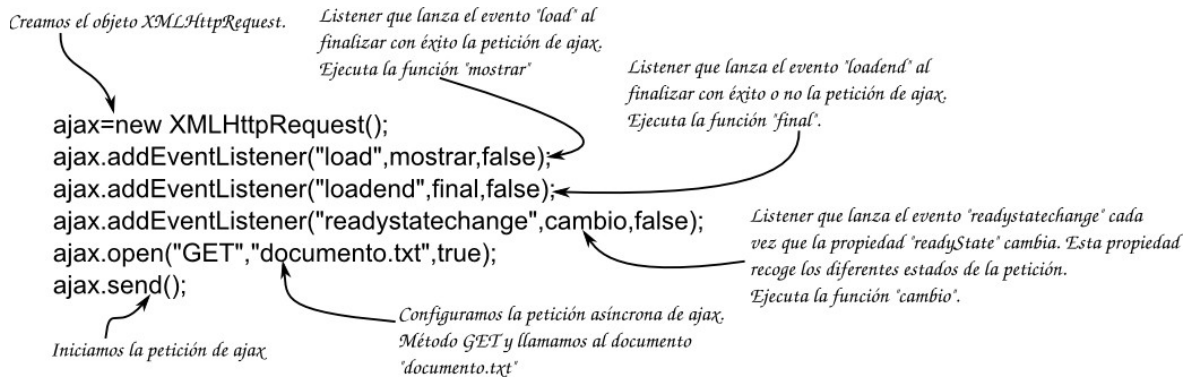
- responseXML: Propiedad que retorna la respuesta de la solicitud en formato XML.

Veamos un primer código donde trabajamos con una respuesta de texto.

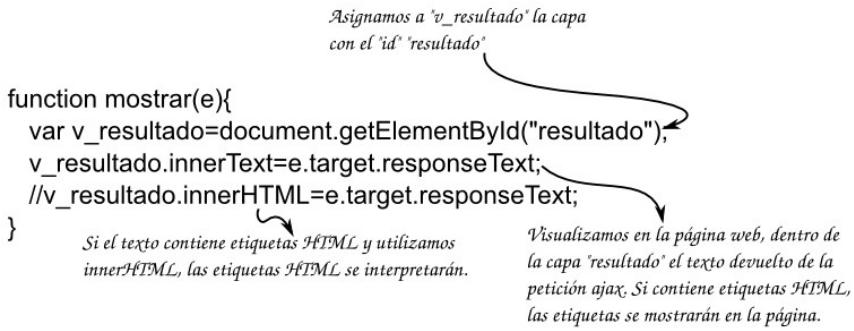
```
<!DOCTYPE html>
<html lang="es">
<head>
<title>ajax2-ejemplo01</title>
<script>
var ajax;
window.addEventListener("load",inicia,false)

function inicia(){
  var obj=document.getElementById("boton");
  obj.addEventListener("click",solAjax,false);
}
```

```
function solAjax(){
  ajax=new XMLHttpRequest();
  ajax.addEventListener("load",mostrar,false);
  ajax.addEventListener("loadend",final,false);
  ajax.addEventListener("readystatechange",cambio,false);
  ajax.open("GET","documento.txt",true);
  ajax.send();
}

function mostrar(e){
  var v_resultado=document.getElementById("resultado");
  v_resultado.innerText=e.target.responseText;
}

function cambio(e){
  console.log(ajax.readyState);// variable pública
  console.log(e.target.readyState); // utilizar el objeto event.
}
function final(){
  alert("todo finalizado");
}
</script>

</head>
<body>
<section id="peticion">
<button id="boton">Pulsar</button>
</section>
<section id="resultado">
</section>
</body>
</html>
```

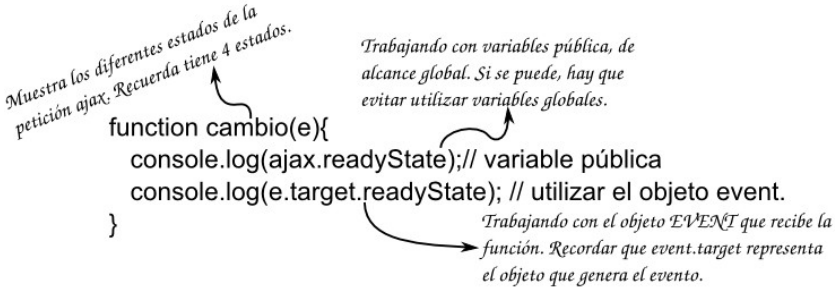
Función “solAjax”



Función “mostrar”



Función “cambio”



Explicaremos ahora, un poco más detallado, el funcionamiento del evento “progress”. Las tres propiedades que tenemos que estudiar para ver el progreso de descarga o de carga de archivos son:

- lengthComputable: Esta propiedad nos indica si podemos o no realizar el cálculo del progreso. Devuelve “verdadero” cuando se puede realizar el cálculo y “false” cuando no se puede realizar el cálculo.

- loaded: Esta propiedad retorna el total de byte que se han enviado o se han recibido de la petición ajax.

- total: Propiedad que retorna el total de byte que se van a transmitir en la petición ajax.

Veamos ahora un pequeño ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>ajax2-ejemplo02</title>
<script>
var ajax;
window.addEventListener("load",inicia,false)

function inicia(){
  var obj=document.getElementById("boton");
  obj.addEventListener("click",solAjax,false);
}
```

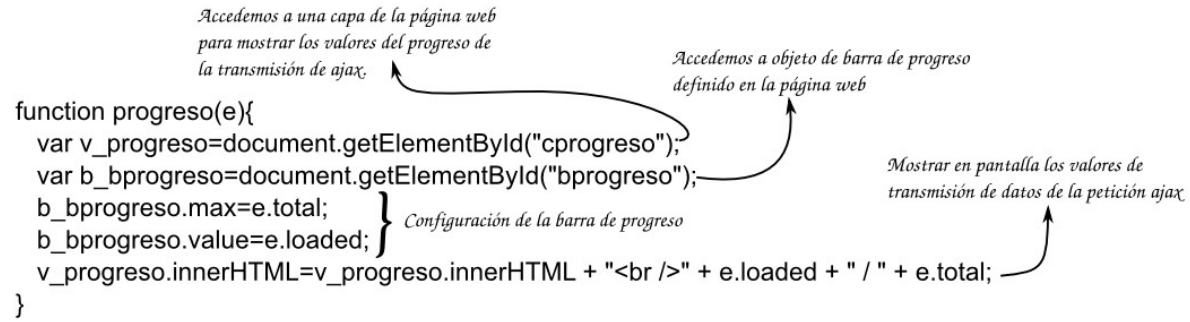
```
function solAjax(){
  ajax=new XMLHttpRequest();
  ajax.addEventListener("load",mostrar,false);
  ajax.addEventListener("progress",progreso,false);
  ajax.open("GET","documento2.txt",true);
  ajax.send();
}

function mostrar(e){
  var v_resultado=document.getElementById("resultado");
  v_resultado.innerText=e.target.responseText;
}
function progreso(e){
  var v_progreso=document.getElementById("cprogreso");
  var b_bprogreso=document.getElementById("bprogreso");
  b_bprogreso.value=e.loaded;
  b_bprogreso.max=e.total;
  v_progreso.innerHTML=v_progreso.innerHTML + "<br />" + e.loaded + " / " + e.total;
}
</script>

</head>
<body>
<section id="peticion">
<button id="boton">Pulsar</button>
</section>
<section id="cprogreso">
<progress id="bprogreso" value="0" max="100"></progress>
</section>
<section id="resultado">
</section>
</body>
</html>
```

Función solAjax

```
ajax=new XMLHttpRequest();
ajax.addEventListener("load",mostrar,false);
ajax.addEventListener("progress",progreso,false);
ajax.open("GET","documento2.txt",true);
ajax.send();
```

Listener que se lanzará durante la petición ajax, Se ejecutará cada cierto tiempo para poder controlar el progreso de transmisión de datos.

Función “progreso”



Enviar datos

Para enviar datos a un servidor siempre se utilizaba un formulario. En html5 existe el objeto **FormData**, que nos permite crear formularios al vuelo (ficticios) para luego poder enviarlo en una petición ajax.

El constructor del objeto FormData es **FormData()**,

var datos=new FormData()

Métodos

- append(nombre,valor): Método que utilizamos para añadir los pares “campo/valor” del formulario. Si existe el campo añade el valor al campo.

  formdata.append(“campo1”,”valor1”)

  Ahora veamos un ejemplo para crear un objeto FormData a partir de un formulario

  miform=document.getElementById(“formu”);
  formdata=new FormData(miform);

- delete(): Método que elimina un par “campo/valor”.

  delete(“campo”) → borra la entrada “campo”.

- entries(): Método que devuelve un iterador. Por cada iterador devuelve un array donde el primer elemento contiene el nombre del campo y el segundo elemento contiene el valor del campo.

  for (a of formdata.entries()9{
  console.log(a);
  }

  visualiza

  [“campo1”,”valor1”]
  [“campo2”,”valor2”]
  ...

- get(): Devuelve el primer valor del campo pasado como argumento

  formdata.get("campo")

  visualiza

  "valor"

- getAll(): Método que devuelve un array con todos los valores asignados al campo pasado como argumento. Recuerde el funcionamiento del método "append".

  formdata.getAll("campo")

  visualiza

  ["valor1","valor2"]

- has(): Método que devuelve un valor boleano. Si existe el campo pasado como argumento devuelve "verdadero" y si no existe devuelve "falso".

  formdata.has("campo1") → true
  formdata.has("campono") → false

- keys(): Este método devuelve un iterador que utilizaremos para recorrer todos los nombres de campos del formulario.

```
for (a of formdata.Keys(){
    console.log(a)
}
```

  visualiza

  "campo1"
  "campo2"
  ….....

- set(): Método que recibe como argumento un par "campo/valor". Si existe el campo borra todas las entradas de dicho campo y crea una entrada nueva con ese "campo" y ese "valor". Si no existe crea un nuevo par dentro del FormData.

  Formdata.set("campo1","valor")

- values(): Método que funciona igual que el método "keys", pero ahora devuelve los valores del FormData.

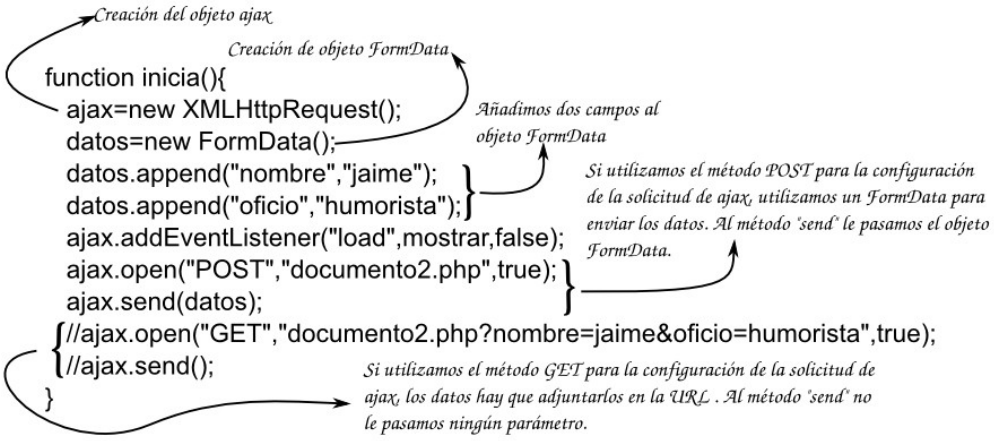
- forEach(): Método que recibe como parámetro una función. Esta función recibe, a su vez, tres parámetros. Estos parámetros son: campo, valor, objeto

```
formdata.forEach(function(campo,valor,objeto){
  console.log(campo);  // campo1
  console.log(valor); // valor1
  console.log(objeto); // formdata
});
```

El único método que funciona en la mayoría de navegadores actuales es "append" los otros métodos funciona solamente con las versiones de los navegadores más reciente.

Veamos ahora un pequeño ejemplo donde mostraremos como mandar datos al servidor.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>ajax2-ejemplo03</title>
<script>
window.addEventListener("click",inicia,false)
function inicia(){
  ajax=new XMLHttpRequest();
  datos=new FormData();
  datos.append("nombre","jaime");
  datos.append("oficio","humorista");
  ajax.addEventListener("load",mostrar,false);
  ajax.open("POST","documento2.php",true);
  ajax.send(datos);
  //ajax.open("GET","documento2.php?nombre=jaime&oficio=humorista",true);
  //ajax.send();
}

function mostrar(e){
  var v_resultado=document.getElementById("resultado");
  v_resultado.innerHTML=e.target.responseText;
}

</script>

</head>
<body>
<section id="peticion">
<button id="boton">Pulsar</button>
</section>
<section id="resultado">
</section>
</body>
</html>
```



Y en el servidor tendremos el siguiente documento...

```
<?php
  if(!empty($_POST)){
    echo '$_POST    '."<br>";
    echo "Tu nombre es " . $_POST["nombre"]."<br>";
    echo "y eres " . $_POST["oficio"];
  } elseif( !empty($_GET)) {
    echo '$_GET     '."<br>";
    echo "Tu nombre es " . $_GET["nombre"]."<br>";
    echo "y eres " . $_GET["oficio"];
    }
?>
```

Recibir datos

Ahora estudiaremos como recibir datos de diferentes tipo. Estudiaremos como recibir datos de tipo texto, xml y json.

Recibir datos de tipo texto

Como hemos visto en apartados anteriores, ajax tiene una propiedad que recibe los datos de tipo texto. Esta propiedad es *responseText*. También podemos trabajar con la propiedad genérica *response*. Esta propiedad recibirá los datos según el valor de la propiedad *responseType*. En este caso asignaremos el valor “text” a la propiedad *responseType*.

Las propiedades *response* y *responseText* tendrán el mismo valor.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>ajax2-ejemplo04</title>
  <script>
    window.addEventListener("click",inicia,false)
    function inicia(){
      ajax=new XMLHttpRequest();
      datos=new FormData();
      datos.append("nombre","jaime");
      datos.append("oficio","humorista");
```

```
        ajax.addEventListener("load",mostrar,false);
        ajax.responseType="text";
        ajax.open("POST","documento2.php",true);
        ajax.send(datos);
     }
     function mostrar(e){
        var v_resultado=document.getElementById("resultado");
        v_resultado.innerHTML=e.target.response;
        //v_resultado.innerHTML=e.target.responseText;
     }
  </script>
</head>
<body>
  <section id="peticion">
    <button id="boton">Pulsar</button>
  </section>
  <section id="resultado">
  </section>
</body>
</html>
```

```
function inicia(){
    ajax=new XMLHttpRequest();
    datos=new FormData();
    datos.append("nombre","jaime");
    datos.append("oficio","humorista");
    ajax.addEventListener("load",mostrar,false);
    ajax.responseType="text";
    ajax.open("POST","documento2.php",true);
    ajax.send(datos);
}
function mostrar(e){
    var v_resultado=document.getElementById("resultado");
    v_resultado.innerHTML=e.target.response;
    //v_resultado.innerHTML=e.target.responseText;
}
```

Propiedad para indicar que tipo de dato será devuelto por la petición ajax

La propiedad response == responseText

Si intentamos acceder a *responseXML* javascript nos dará un error de acceso a la propiedad.

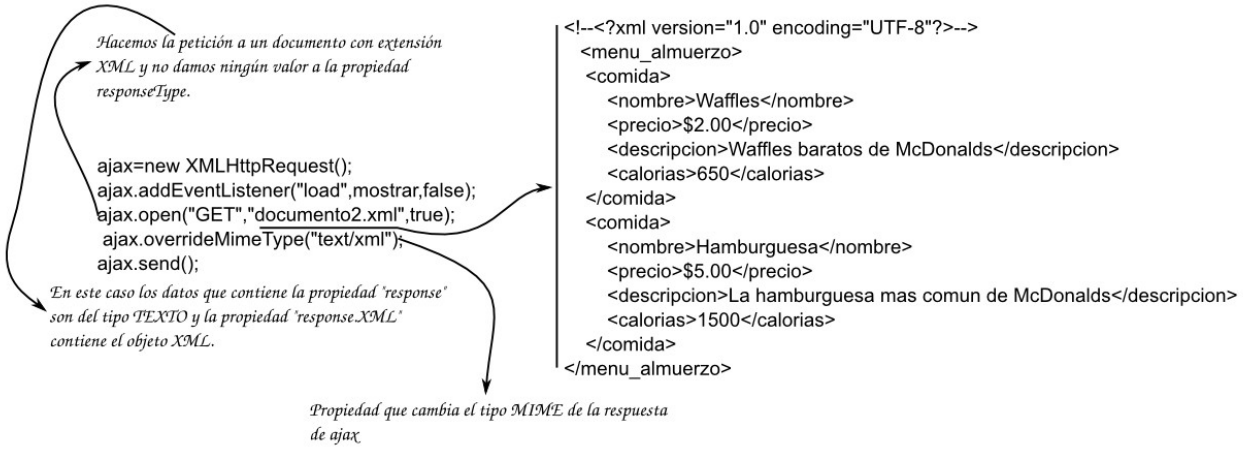
Recibir datos del tipo document o XML

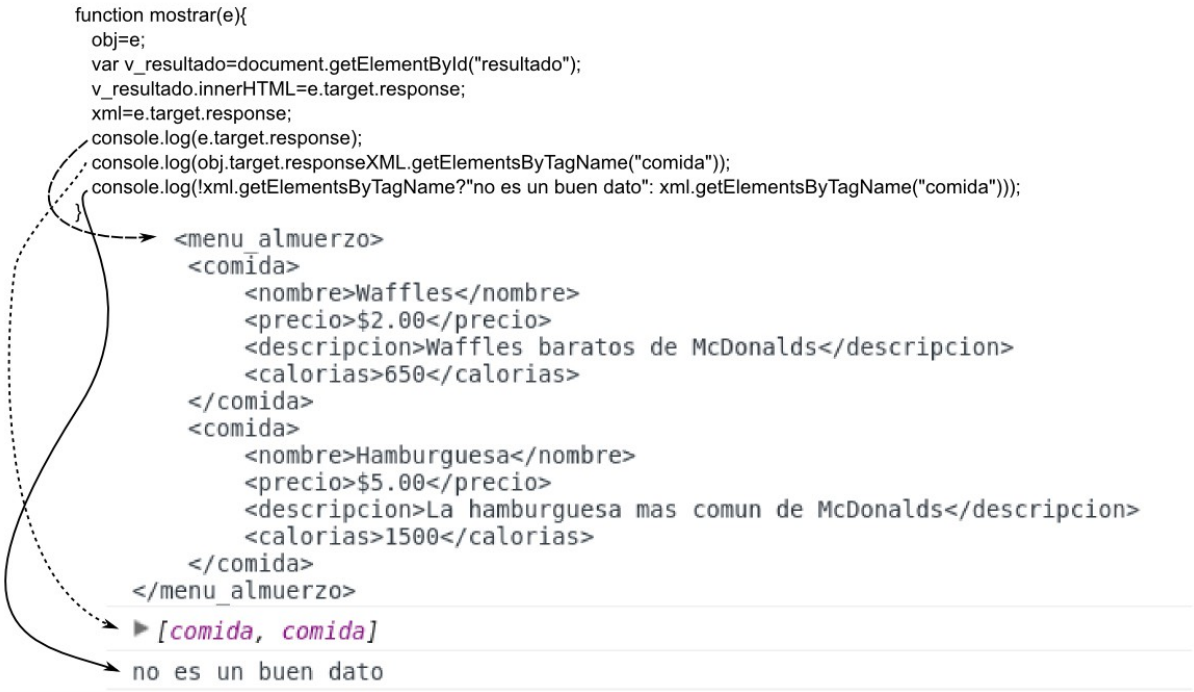
Para recibir datos de tipo *document* o *XML* tenemos que asignar el valor “document” a la propiedad *responseType*. Dependiento a que tipo de fichero llamemos o como nos devuelva los datos podremos recibir un *HTMLdocument o un XMLdocument*.

Un objeto *HTMLdocument* es un objeto que tiene la estructura de una página web y los datos están dentro de la etiqueta *body*.

Se pueden dar diferentes situaciones para acceder a los datos de tipo XML.

En este primer caso vamos a acceder a un documento xml bien formado. No declaramos la propiedad *responseType* de ajax y la propiedad *overrideMimeType*, que se utiliza para cambiar el tipo MIME del documento recibido, no tiene ninguna función sobre el resultado.





Por tanto, reponseXML tiene el objeto XML, response es texto y la propiedad overrideMimeType no hace absolutamente nada.

En el siguiente ejemplo vamos a usar la propiedad *responseType*.

```
function inicia(){
  ajax=new XMLHttpRequest();

  ajax.addEventListener("load",mostrar,false);
  //ajax.open("GET","documento2.php",true);
  ajax.open("GET","documento2.xml",true);
  ajax.send();
  ajax.responseType="document";
 // ajax.overrideMimeType("text/xml");
}

function mostrar(e){
  obj=e;
  var v_resultado=document.getElementById("resultado");
  v_resultado.innerHTML=e.target.response;
  xml=e.target.response;
  console.log(e.target.response);
  console.log(obj.target.responseXML.getElementsByTagName("comida"));
  console.log(!xml.getElementsByTagName?console.log("no es un buen dato"): console.log(xml.getElementsByTagName("comida")));
}
```

Importante !!!!
Estamos haciendo la petición a un documento xml bien formado

Esta línea la podemos comentar o no, NO tiene ningún efecto sobre el resultado final

Se objttiene la misma respuesta, un objeto XML

Ahora realizaremos la petición a un fichero php, como el siguiente:

```
<?php
//URL del artículo: http://www.ejemplode.com/21-xml/525-ejemplo_de_menu_de_comidas_en_xml.html
//Fuente: ejemplos de Menú de comidas en XML
// <?xml version="1.0" encoding="UTF-8" \?\>
  $pp=<<<XML
  <menu_almuerzo>
   <comida>
      <nombre>Waffles</nombre>
      <precio>$2.00</precio>
      <descripcion>Waffles baratos de McDonalds</descripcion>
      <calorias>650</calorias>
   </comida>
   <comida>
      <nombre>Hamburguesa</nombre>
      <precio>$5.00</precio>
      <descripcion>La hamburguesa mas comun de McDonalds</descripcion>
      <calorias>1500</calorias>
   </comida>
</menu_almuerzo>
XML;
header("Content-Type: text/xml");
echo $pp;
?>
```

No incluir esta líneas. Si comentamos esta línea es importante los barras invertidas delante de la interrogación y del mayor que. De lo contrario lo toma como final de código de php.

Importante para indicar que devuelve un objeto XML de lo contrario devolvera un texto. Si omitimos esta línea es importante incluir en el código de javascript la definición de la propiedad overrideMimeType('text/xml')

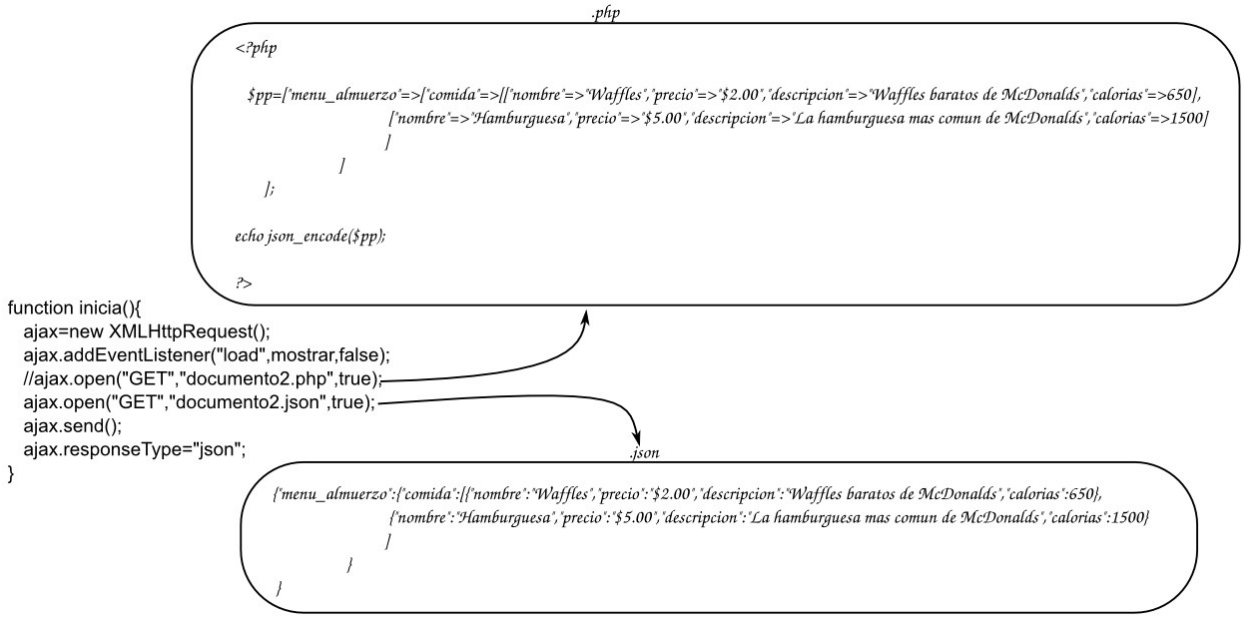
Situaciones que se pueden dar:

responseType="" → response es texto
responseXML es un documento xml

responseType="document" → response es documento xml
responseXML es documento xml

Recibir datos json

Recibir un objeto json es muy sencillo. Solo tenemos que asignar el valor "json" a la propiedad *responseType* (responseType="json") y podemos llamar a un fichero con extensión "json", donde, habrá un objeto json bien definido o llamar a un fichero php donde se devolverá un array asociativo convertido a objeto json.



Subir archivos

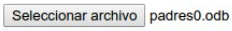
Subir archivos

Veamos como subir archivos de un modo sencillo. Podemos subir un único fichero o varios ficheros.

Antes de empezar a estudiar como subir los ficheros tenemos que ver primero la etiqueta *input* con *type="file"* y como maneja los datos que recibe.

Entre los atributos que tiene esta etiqueta estudiaremos los siguientes:

type="file" → con este atributo indicamos que vamos a seleccionar archivos locales para subirlos al servidor. Cuando añadimos este atributo se añade un botón, de forma automática, que nos abre un examinador de dispositivos local, y de este modo podemos seleccionar el fichero o los ficheros que queremos subir.



accept → indica que tipo de fichero se podrá enviar al servidor. Se puede especificar más de un valor separados por coma. Si no incluimos este atributo podemos seleccionar cualquier tipo de fichero. Los valores que podemos asignar son:

la extensión de un fichero: .jpg, .png ……. No se nos olvide el punto.

audio/*: Todos los ficheros de audio.

video/*: Todos los ficheros de video.

image/*: Todos los ficheros de image.

Tipo MIME: Un valor válido de tipo MIME. Podemos consultar esta lista en http://www.iana.org/assignments/media-types/media-types.xhtml .

multiple → Se incluye cuando queremos subir más de un fichero. A este atributo le podemos asignar el valor true o false. Si queremos asignarle el valor true basta con incluir el atributo y no asignarle ningún valor.

```
<input type=”file” multiple >
```

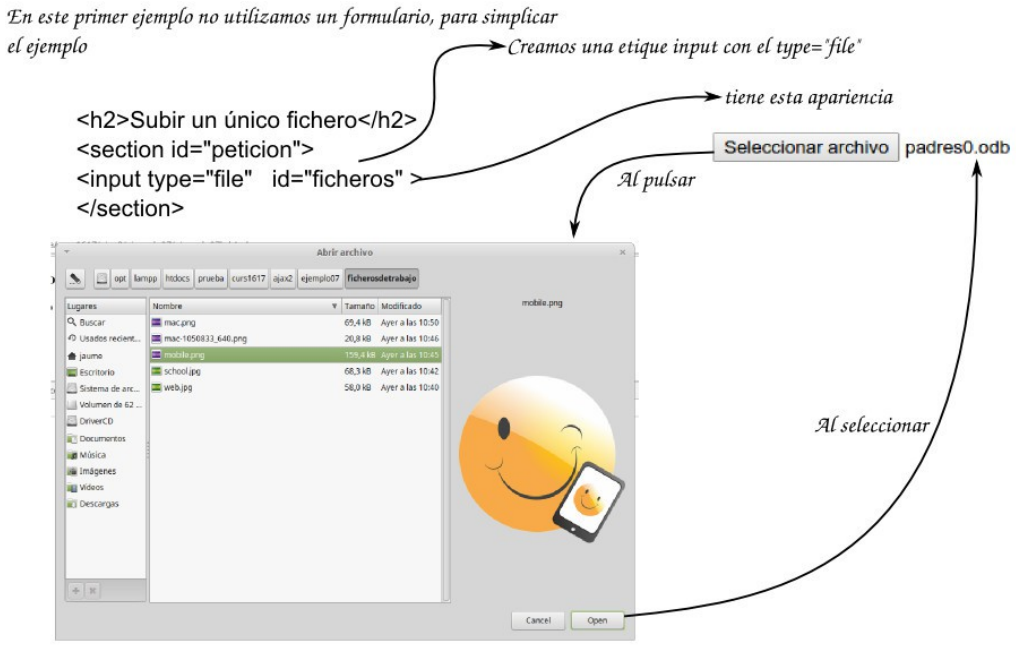
Para poder estudiar y ampliar información sobre los atributos de esta etiqueta podemos consultar la página http://www.w3schools.com/jsref/dom_obj_fileupload.asp

Repasamos el atributo *enctype* de la etiqueta *form*. Este atributo se utiliza para indicar al navegador como tiene que enviar la información al servidor. Por omisión, este atributo tiene el valor *application/x-www-form-urlencoded* que codifica los nombres de control y los valores. No es el modo ideal para enviar ficheros. El otro valor que utilizamos es *multipart/form-data*, este es el valor que asignaremos. Con este valor no codificamos los caracteres de control.

Para aprender un poco más sobre el atributo *enctype* consultar el siguiente enlace http://html.conclase.net/w3c/html401-es/interact/forms.html#h-17.13.4

Ahora tenemos que diferenciar entre subir un único archivo o subir varios archivos. Empecemos estudiando como subir un archivo y luego realizaremos los cambios necesarios para poder subir varios archivos.

Código html



Código javascript

```
window.addEventListener("load",inicia,false);
var datos=new FormData();
function inicia(){
   var cficheros;
   cficheros=document.getElementById("ficheros");
   cficheros.addEventListener("change",comprueba);
}

function comprueba(e){
   datos.append("mficheros",e.target.files[0]);
   miajax();
}

function miajax(){
   ajax=new XMLHttpRequest();
   ajax.addEventListener("load",mostrar,false);
   ajax.open("POST","documento2b.php",true);
   ajax.send(datos);
}

function mostrar(e){
   var v_resultado=document.getElementById("resultado");
   v_resultado.innerHTML=e.target.responseText;
   xml=e.target
}
```

Creamos el formData para enviar al servidor

Al cambiar el valor del campo se lanza todo el proceso de subida de fichero

Quien produce el evento es campo. Este objeto devuelve un listFile, es decir, un objeto que contieno una lista con los ficheros que se van a subir al servidor.

Añadimos el campo al formData

Realizamos el envío al servidor

Codigo php

```
$fichero = $_FILES;
define("SALT","<br>");
echo SALT.SALT.SALT;
if(move_uploaded_file($fichero["mficheros"]["tmp_name"],"./".$fichero["mficheros"]["name"])){
        echo "fichero ".$fichero["mficheros"]["name"] ." subido correctamente ".SALT;
} else {
        echo "hay error ".count($fichero). SALT;
}
```

Array global de php que utiliza para recoges los ficheros subidos y su parámetros

Atributo "name" del campo. En este caso es el nombre que le hemos dado en el formData.

Método de php para grabar un fichero subido desde el cliente.

Vamos a estudiar las modificaciones que tenemos que realizar en el código para poder subir varios archivos.

En el código html hacemos la siguiente modificación

```
<section id="peticion">
   <h2>Subir múltiples ficheros</h2>
   <input type="file" multiple id="ficheros" >
 </section>
```

Se añade el atributo "multiple" a la etiqueta

En el código javaScript

```
function comprueba(e){
    for(a=0;a<e.target.files.length;a++)
      datos.append("mficheros[]",e.target.files[a]);
    miajax();
  }
```

Al nombre del campo le añadimos unos corchetes para crear un array.

Y en php

Ahora tenemos que trabajar con unos índices en el array global que recoge los ficheros que suben al servidor

```
<?php
 $fichero = $_FILES;
 define("SALT","<br>");
 echo SALT.SALT.SALT;
 for($a=0;$a<count($fichero["mficheros"]["name"]);$a++){
    if(move_uploaded_file($fichero["mficheros"]["tmp_name"][$a],"./".$fichero["mficheros"]["name"][$a])){
          echo "fichero ".$fichero["mficheros"]["name"][$a] ." subido correctamente".SALT;
    } else {
          echo "hay error ".count($fichero).SALT;
    }
 }
?>
```

Veamos un último ejemplo de como subir ficheros al servidor. Ahora trabajaremos con un formulario y crearemos el formData a partir de dicho formulario. El formulario está formado por un campo texto y un campo *file*.

Modificaciones en html

```
<form id="formulario" enctype="multipart/form-data" >
      <input type="text" id="texto", name="ntexto" >
      <input type="file" name="mficheros" id="ficheros" >
      <input type="submit" value="enviar" >
</form>
```

En javaScript solo cambia lo siguientes

```
function comprueba(e){
    datos=new FormData(benviar);
    miajax();
    e.preventDefault();
}
```

Creamo el formData a partir del formulario benviar=document.getElementById("formulario")

En php

Campo texto

El tratamiento del fichero no varía

```
<?php
 $datos= $_POST["ntexto"];
 $fichero = $_FILES;
 define("SALT","<br>");
 if(move_uploaded_file($fichero["mficheros"]["tmp_name"],"./".$fichero["mficheros"]["name"])){
       echo "Se ha subido el fichero y se ha leído el texto " . $datos . SALT;
 } else {
        echo "hay error ".count($fichero).SALT;
 }
?>
```

Bajar archivos

Igual que hemos hecho antes tenemos que explicar primero unos conceptos previos.

URL → Este interfaz representa un objeto que proporciona métodos estáticos utilizados para la creación de objetos de enlace.

URL.createObjectURL(blob) → Es un método estático que devuelve un cadena (DOMString) que contiene una URL que representa el parámetro.

URL.revokeObjectURL() → Es un método estático que libera un objeto URL creado previamente con URL.createObjecgtURL.

Petición directa a un recurso en el servidor

Código javascript

```
function miajax(){
  ajax=new XMLHttpRequest();
  ajax.addEventListener("load",mostrar,false);
  ajax.open("GET","ficherosdetrabajo/mac.png",true);
  ajax.send();
  ajax.responseType="blob";
}

function mostrar(e){
  var v_resultado=document.getElementById("resultado");
  var imagen=document.createElement("img");
  imagen.src=URL.createObjectURL(e.target.response);
  v_resultado.appendChild(imagen);
}
```

Petición directa a un recurso en el servidor

Creamos la etiqueta img

Creamos el objeto enlace y se lo asignamos al atributo src de la etiqueta img

No hay código en el servidor.

Petición a una página php que devuelve un recurso del servidor

Código javaScript

```
function miajax(){
  ajax=new XMLHttpRequest();
  ajax.addEventListener("load",mostrar,false);
  ajax.open("GET","ficherosdetrabajo/mac.png",true);
  ajax.send();
  ajax.responseType="blob";
}

function mostrar(e){
  var v_resultado=document.getElementById("resultado");
  var imagen=document.createElement("img");
  imagen.src=URL.createObjectURL(e.target.response);
  v_resultado.appendChild(imagen);
}
```

Petición directa a un recurso en el servidor

Importante

Creamos la etiqueta img

Creamos el objeto enlace y se lo asignamos al atributo src de la etiqueta img

Código php

```
<?php
    $fichero="ficherosdetrabajo/school.jpg";
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($fichero).'"');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . filesize($fichero));
    readfile($fichero);
?>
```

No es obligatorio. Es buena constumbre enviar la cabecera de la página web correctamente

Escribe el contenido del fichero que recibe como parámetro

Espero que con esta información quede claro el propósito de estos apuntes.