C.F.G.S.: DESARROLLO DE APLICACIONES WEB
Módulo: DESARROLLO WEB EN ENTORNO CLIENTE

02 AJAX Y JQUERY

AJAX Y JQUERY

https://uniwebsidad.com/libros/fundamentos-jquery/capitulo-7/metodos-ajax-de-jquery?from=librosweb

jQuery posee varios métodos para trabajar con Ajax, todos ellos basados en el método $.ajax().

$.ajax()

El método $.ajax() es configurado a través de un objeto que tiene todas las instrucciones que necesita jQuery para completar la petición. Se pueden especificar las acciones a realizar en caso de que la petición haya fallado o no.

http://api.jquery.com/jQuery.ajax/

Por ejemplo:

```
$.ajax({
  url: '/ruta/hasta/pagina.php',
  data: 'parametro1=valor1&parametro2=valor2',
  type: 'POST',
  dataType:txt,
  async: true,
  success: procesaRespuesta,
  beforeSend:antesEnvío,
  error: muestraError,
  complete:trasPeticion
});
```

- **url:** URL para la petición. Es parámetro obligatorio
- **data**: La información a enviar. Esta puede ser tanto
  - un objeto (por ejemplo { c1 : val1, c2 : val2 }
  - como una cadena de datos (por ej. 'parametro1=valor1&parametro2=valor2')
- **type:** Especifica si será una petición POST o GET

- **async**: Indica si la petición es asíncrona. No es obligatorio, su valor por defecto es true, el habitual para las peticiones AJAX.
- **datatype**: El tipo de dato que se espera como respuesta. Si no se indica ningún valor, jQuery lo deduce a partir de las cabeceras de la respuesta. Los posibles valores son:
    - xml (se devuelve un documento XML correspondiente al valor responseXML)
    - html (devuelve directamente la respuesta del servidor mediante el valor responseText)
    - script (se evalúa la respuesta como si fuera JavaScript, ejecuta el js)
    - json (se evalúa la respuesta como si fuera JSON y se devuelve el objeto JavaScript generado)
    - txt (devuelve directamente la respuesta del servidor mediante el valor responseText)

- **success**: Permite establecer la función que se ejecuta cuando una petición se ha completado de forma correcta. La función recibe como primer parámetro los datos recibidos del servidor, previamente formateados según se especifique en la opción dataType
- **beforeSend**: establece una función que se ejecutará antes de enviar la petición
- **error**: Código a ejecutar si la petición falla; son pasados como argumentos a la función el objeto jqXHR (extensión de XMLHttpRequest), un texto con el estatus de la petición(timeout, error, abort, o parsererror) y un texto con la descripción del error que haya dado el servidor(Not Found o Internal Server Error)
- **complete**: Código a ejecutar cuando la petición se ha completado, sin importar si falló o no. La función recibe como argumentos el objeto jqXHR y un texto especificando el estatus de la misma petición (success, notmodified, error, timeout, abort, o parsererror).

Ver ***Ejemplo 1 XML jQuery***
Ejemplo 2 txt jQuery
Ejemplo 3 JSON jQuery
Ejemplo 4 js jQuery

OTROS MÉTODOS

load()

Este método es el más sencillo de todos y sirve para cargar documentos HTML dentro de cada uno de los elementos del objeto jQuery al que se aplica (obtiene el código HTML de una URL y rellena los elementos seleccionados con la información obtenida)

Este método se puede ejecutar sobre cualquier objeto jQuery y cargará en los elementos de ese objeto la respuesta de la petición realizada al servidor considerándola código HTML si la petición se efectúa correctamente (si no, no hace nada).

La sintaxis de este método es la siguiente:

load(*url*,[*datosPetición*],[*referenciaFunciónCompletada(responseText,textStatus,XHR)*])

Podemos escribir a continuación del argumento url, separado por un espacio, un selector CSS que identifique qué parte de la respuesta es la que queremos cargar en el interior de los elementos del objeto jQuery.

```
$('#respuesta').load('respuesta.html #jquery',gestionarRespuesta);
```

Este método efectúa una petición de tipo GET si se omiten los datos de la petición o si se aportan como una cadena de caracteres de paso de parámetros (pares datos valor separados por signos &), y de tipo POST si datosPetición se aporta en forma de objeto.

El argumento *referenciaFunciónCompletada* corresponde a una función que recibe como argumentos:

- la propiedad XMLHttpRequest.responseText (contedrá el código del archivo html)
- el valor textStatus
- una referencia al propio objeto XMLHttpRequest

Ver Ejemplo 5 load

$.get()

Esta función global sirve para realizar peticiones GET, cargar la respuesta (que puede ser de tipo HTML, XML, JSON o Script), y ejecutar una función cuando la petición se ha completado. Esta función, como todas las siguientes, devuelve un objeto de jQuery de tipo jqXHR, que básicamente es el mismo objeto XMLHttpRequest que ya conocemos del DOM (con sus mismas propiedades y métodos, como status, timeout, responseText, getResponseHeader(), …), pero generalizado para adaptarse también a las peticiones de datos de tipo JSON y script.

Su sintaxis es la siguiente:

$.get(*url*,[*datosPetición*],[*gestionarExito(datosRecibidos,textStatus,jqXHR)*],
[*tipoDatosRecibidos*])

Los datos de la petición pueden expresarse como una cadena codificada en formato url, o como un objeto.

El argumento *tipoDatosRecibidos* acepta los valores 'html', 'xml', 'json' y 'script', y determina de qué tipo son los datos que se esperan del servidor; por defecto, jQuery intentará usar el tipo MIME proporcionado en la respuesta por el servidor.

En el caso de datos XML, la función jQuery nos ofrece otra sintaxis alternativa que admite esos datos XML como argumento, de modo que luego podemos recurrir al método find para localizar elementos determinados del documento XML; por ejemplo: $(respuestaXML).find('nombre');

$.post()

Esta función global es la equivalente a la anterior pero para peticiones de tipo POST.

Su sintaxis es:

$.post(*url*,[*datosPetición*],[*gestionarExito(datosRecibidos,textStatus,jqXHR)*],
[*tipoDatosRecibidos*])

$.getJSON()

Esta función global sirve para realizar peticiones GET que cargan directamente datos JSON. Es exactamente igual que $.get() cuando en esta última se especifica como tipo de datos JSON, es decir, envía los datos de la petición con el método GET y obtiene una cadena expresada en JSON que convierte automáticamente en un objeto que pone a disposición de JavaScript mediante el argumento *datosRecibidos*:

$.getJSON(*url*,[*datosPetición*],[*gestionarExito(datosRecibidos,textStatus,jqXHR)*])

Ver Ejemplo 6 getJSON

$.getScript()

Esta función global sirve para realizar peticiones GET que cargan directamente scripts JavaScript y los ejecutan.

Su sintaxis es la siguiente:

```
$.getScript(url,[gestionarExito(datosRecibidos,textStatus,jqXHR)])
```

Ver Ejemplo 7 getScript

Uno de los argumentos que podemos aportar a cualquiera de los métodos anteriores es una función que se ejecuta al completar la petición. Esta función puede recibir a su vez como argumento una cadena de caracteres que nos permitirá saber cuál ha sido el resultado de la petición consultando su valor, que puede ser uno de los siguientes:

- *'success': Indica que la respuesta se ha recibido correctamente.*
- *'notmodified': Indica que no se ha recibido respuesta porque ya estaba disponible en la caché del navegador.*
- *'error': Indica que se ha producido un error en la petición.*
- *'timeout': Indica que el plazo de la petición ha caducado sin que llegue a concluir.*
- *'abort': Indica que la petición ha sido cancelada.*
- *'parseerror': Indica que se ha producido un error al interpretar la respuesta, generalmente cuando ésta es de tipo XML.*

En el caso de las funciones globales (no del método load) también podemos gestionar el resultado de la petición configurando los métodos

- *done(datosRecibidos,textStatus,jqXHR),*
- *fail(jqXHR,textStatus,textoMensajeError)*
- *always(jqXHR,textStatus)*

del objeto jqXHR que devuelven estas funciones; estos métodos reciben como argumento la referencia a la función que queremos ejecutar en cada uno de estos casos.

```
$.getJSON('enlaces.json',cargarEnlaces)
                                .done(function(){alert('complete');})
                                .fail(function(){alert('error');})
                                .always(function(){alert('success');});
```

Ver Ejemplo 8