*jQuery y Ajax*

Contenido

[1.-Método jQuery load () 1](#_Toc64219474)

[2.-Métodos jQuery get () y post () de AJAX 2](#_Toc64219475)

[3.-La función $.ajax() 3](#_Toc64219476)

[*4.-INFORMACIÓN GENERAL JQUERY* 5](#_Toc64219477)

[4.1.-Eventos en jQuery 5](#_Toc64219478)

[4.2.- Selectores de jQuery básicos 5](#_Toc64219479)

[4.3.- Métodos y funciones JQuery 6](#_Toc64219480)

[5.- Lectura de JSON Y XML con $.ajax 7](#_Toc64219481)

# 1.-Método jQuery load ()

El método load() de jQuery es un método AJAX simple pero poderoso. Carga datos de un servidor y coloca los datos devueltos en el elemento seleccionado.

**Sintaxis:**

$(*selector*).load(*URL,data,callback*);

El parámetro de URL requerido especifica la URL que desea cargar.

El parámetro data es opcional y especifica un conjunto de pares clave / valor de cadena de consulta para enviar junto con la solicitud.

La función callback es opcional, y es ahí donde pondremos la función de retorno, que será llamada una vez terminada la petición. En esa función realizaremos tareas adicionales, ya que la acción por defecto de cargar en un objeto el contenido devuelto en la petición, la realiza el propio método load().

Ejemplo:

$("#div1").load("demo\_test.txt");

También es posible agregar un selector de jQuery al parámetro de URL:

Ejemplo:

$("#div1").load("demo\_test.txt #p1");

El ejemplo anterior carga el contenido del elemento con id = "p1", dentro del archivo "demo\_test.txt", en un <div>elemento específico :

# 2.-Métodos jQuery get () y post () de AJAX

La función $.post()

Nos permite realizar peticiones AJAX al servidor, empleando el método POST. Su sintaxis es la siguiente:

$.post( url, [datos], [callback], [tipo] )

Ejemplos:

$.post("test.php");

$.post("test.php", { nombre: "Juana", hora: "11am" } );

$.post("test.php", function(resultados) {

alert("Datos Cargados: " + resultados);

});

[Más información sobre el método .post() en jQuery.](http://api.jquery.com/jQuery.post/)

La función $.get() y $.getJSON()

Hacen prácticamente lo mismo que POST, y tienen los mismos parámetros, pero usan el método GET para enviar los datos al servidor. Si recibimos los datos en formato JSON, podemos emplear $.getJSON() en su lugar.

$.get( url, [datos], [callback], [tipo] )

$.get JSON ( url, [datos], [callback], [tipo] )

[Más información sobre el método .get() en jQuery.](http://api.jquery.com/jQuery.get/)

# 3.-La función $.ajax()

Hasta ahora hemos visto que jQuery nos provee varias formas de recuperar y enviar datos al servidor utilizando ajax: Método load(), Funciones $.get, $.post y $.getJSON.

Las anteriores son versiones simplificadas y especializadas de $.ajax que es la más completa de todas y permite hacer lo mismo que todas las anteriores pero con la desventaja de es más compleja en su empleo.

El método $.ajax() cuenta con dos sintaxis posibles:

$.ajax(url, { objeto-configurable } );

// o

$.ajax( { objeto-configurable } );

Ejemplo

$.ajax({

url: '/ruta/hasta/pagina.php',

type: 'POST',

async: true,

data: 'parametro1=valor1&parametro2=valor2',

success: procesaRespuesta,

error: muestraError

});

El objeto configurable contendrá uno o varios de los parámetros siguientes:

type : tipo de la petición, GET o POST (GET por defecto).

url : dirección a la que se envía la petición.

data : datos a enviar al servidor.

dataType : tipo de datos que esperas obtener del servidor (si no se especifica, jQuery intenta averiguar de qué tipo se trata).

success : función que se ejecuta cuando se obtiene una respuesta con éxito.

error : función que se llama si la petición no tiene éxito.

Hay muchas otras opciones que pueden ser pasadas como parámetro del objeto. Puedes encontrar la lista completa en la documentación oficial de jQuery.

Una petición get a nuestro servidor puede hacerse de la siguiente forma:

$.ajax({

url: 'https://reqres.in/api/users'

});

[Información ampliada](https://openclassrooms.com/en/courses/4309491-simplifica-tus-proyectos-con-jquery/4981961-el-metodo-ajax)

Sugerencias de estructuras básica con $.ajax() que permita manejo de errores:

$.ajax({

url: "XXXX",

data: 'parametro1=valor1&parametro2=valor2',

type: 'post',

}).done( function() {

alert( 'Success!!' );

}).fail( function() {

alert( 'Error!!' );

}).always( function() {

alert( 'Always' );

});

$.ajax({

url: "pagina1.php",

data: 'parametro1=valor1&parametro2=valor2',

beforeSend: Función mientras se realiza el envío,

success: Función para procesar la llegada de datos exitosa

error: Función si se producen problemas.

});

# *4.-INFORMACIÓN GENERAL JQUERY*

## 4.1.-Eventos en jQuery

jQuery provee métodos para asociar controladores de eventos (en inglés event handlers) a selectores. Cuando un evento ocurre, la función provista es ejecutada. Dentro de la función, la palabra clave this hace referencia al elemento en que el evento ocurre.

Para más detalles sobre los eventos en jQuery, puede consultar <http://api.jquery.com/category/events/>.

**Vincular un evento utilizando un método reducido**

$('p').click(function() {

console.log('click');

});

**Vincular un evento utilizando el método $.fn.on**

$('p').on('click', function() {

console.log('click');

});

Ambas formas son equivalentes y en ambos casos la función se ejecutará tras producirse el evento click sobre el párrafo.

## 

## 4.2.- Selectores de jQuery básicos

Recordemos los más básicos y utilizados, estos son los que intereactuan con ids, clases y etiquetas tal y como se hace en las hojas de estilo CSS.

Con el «.» seleccionamos una clase, con el «#» seleccionamos un id y con el tag de la etiqueta una etiqueta:

/\* Seleccionar etiquetas del DOM \*/

$("elemento")

/\* Seleccionar id del DOM \*/

$("#idelemento")

/\* Seleccionar clase del DOM \*/

$(".claseelemento")

Tal y cómo veis en el código superior podemos seleccionar elementos/etiquetas y elementos con determinados ids o clases.

***Unos ejemplos prácticos y sencillos serían los siguientes:***

/\* seleccionar las etiquetas "p" \*/

$("p")

/\* seleccionar los elementos con id="inicio" \*/

$("#inicio")

/\* seleccionar los elementos con clase="principal" \*/

$(".principal")

/\* seleccionar los elementos con id="inicio" con clase="principal" y que son etiqueta "p" \*/

$("p#inicio.principal")

/\* seleccionar las etiquetas "a" con clase="principal" dentro de una etiqueta "p" \*/

$("p a.principal")

/\* seleccionar los elementos con id="inicio" o clase ="principal" \*/

$("#inicio,.principal")

/\* seleccionar todos los elementos \*/

$("\*")

/\* seleccionar todas las etiquetas "p", "a" y "div" \*/

$("p,a,div")

## 4.3.- Métodos y funciones JQuery

[Ver tablas resumen.](https://norfipc.com/codigos/jquery-todos-selectores-eventos-metodos-funciones.php)

[Así funciona serialize en jQuery: Para crear objetos literales con todos los campos de un formulario con una única sentencia.](https://www.desarrollolibre.net/blog/jquery/asi-funciona-serialize-en-jquery#.YCkVSuhKjDc)

# 5.- JSON Y XML

A principios de los años 90 surgió el problema de que las maquinas pudieran entenderse entre si. Utilizaban diferentes sistemas operativos y sus programas estaban escritos en diferentes lenguajes de programación. Una de las soluciones fue crear el estándar **XML** y más tarde **JSON**, para a través de ellos poder **intercambiar información entre las máquinas** independientemente del sistema operativo o del lenguaje de programación con el que estuviera hecho el software.

**XML (Extensible Markup Language):**

Si conoces [**HTML**](https://programarfacil.com/blog/programacion/introduccion-a-html5/) te resultara familiar, pero XML es más riguroso que HTML. Todas las etiquetas se deben cerrar, cosa que no es obligatorio en HTML y si existe un error en el XML las herramientas que los procesan darán error.



[Leer archivos XML con JQuery](https://www.returngis.net/2011/01/leer-archivos-xml-con-jquery/)

En el siguiente ejemplo, la respuesta que nos consigue $.ajax, “**data**”, ya contienen la información interpretada del XML del servidor y se trata de una lista con los diferentes elementos que contiene y que podemos recorrer utilizando find() para posicionarnos en la etiqueta que nos interesa y con each() para ejecutar la misma función para cada uno de los elementos encontrados.

$.ajax({

type: 'GET',

url: 'agenda.xml',

dataType: 'xml'

}).done((data) =>{

$(data).find('persona').each(function() {

let fila = $('<tr>');

fila.append($(`<td>${$(this).find('id').text()}</td>`));

fila.append($(`<td>${$(this).find('nombre').text()}</td>`));

fila.append($(`<td>${$(this).find('apellido').text()}</td>`));

fila.append($(`<td>${$(this).find('edad').text()}</td>`));

$(“Apuntar al cuerpo de la tabla”).append(fila);

});

**JSON (JavaScript Object Notation):**

Años más tarde salió este formato de intercambio de información más **legible** por el ser humano e igual de eficaz que XML para la comunicación entre maquinas. Está basado en un subconjunto del lenguaje de programación JavaScript. Como el anterior estándar es independiente del lenguaje de programación y del sistema operativo que lo ejecute.

JSON está construido por una colección de **pares de nombre y valor** o por una **lista ordenada de valores**. En muchos lenguajes de programación tenemos elementos parecidos como pueden ser los objetos, una estructura, un diccionario, un array….



JSON Es actualmente el formato más popular para enviar información entre servidor y cliente. La mayoría de las librerías famosas como jQuery ya lidian con el problema de parsear (=interpretar) el jSON de respuesta del servidor y te ofrecen un objeto nativo que puedes usar dentro de tu código.

En el siguiente ejemplo, la respuesta que nos consigue $.ajax, “**data**”, ya contienen la información interpretada del Json del servidor y se trata de una lista con los diferentes elementos que contiene y que podemos recorrer de uno en uno con la función $.each

$.ajax({

type: 'GET',

url: 'agenda.json',

dataType: 'json'

}).done((data) =>{

$.each(data, function (indice, persona){

let fila = $("<tr>");

fila.append($(`<td>${persona.id}</td>`));

fila.append($(`<td>${ persona.nombre}</td>`));

fila.append($(`<td>${ persona.apellido}</td>`));

fila.append($(`<td>${ persona.edad}</td>`));

$(“Apuntar al cuerpo de la tabla”).append(fila);

});

En el ejemplo anterior se utiliza la función each para meter cada elemento obtenido del Json en una fila, poniendo cada campo en una columna.