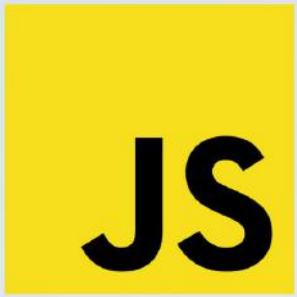
Desarrollo Javascript

del lado del cliente



Miguel Angel Alvarez
Dairo Galeano

desarrolloweb.com/manuales/manual-javascript.html#manual34

Introducción: Desarrollo en Javascript del lado del cliente

En este Manual de Javascript explicamos todos los recursos con los que cuenta un programador para manipulación de una página web. Explicamos cómo un desarrollador, mediante Javascript, puede alterar el estado de una página web para responder a acciones del usuario y crear todo tipo de efectos y aplicaciones web dinámicas.

Básicamente vamos a ver cómo se desarrolla con Javascript del lado del cliente, accediendo a los objetos del navegador por medio de programación de scripts.

En el manual explicaremos los recursos con los que cuentas para modificar la página dinámicamente, gracias a la ejecución de scripts Javascript y la manipulación de los objetos del navegador, ya sea la propia ventana, o los documentos que se están visualizando y todos los objetos que se encuentran en ellos, lo que se conoce como DOM.

Aprenderás también a definir comportamientos como respuesta a eventos del usuario, que es la base de la interacción y que permitirá escribir programas que se ejecutarán cuando ocurren cosas, como clics sobre determinados elementos, salirse de una página, enviar un formulario y un largo etc.

Encuentras este manual online en:
http://desarrolloweb.com/manuales/javascript-lado-cliente.html

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.

Dairo Galeano

Desarrollador independiente

Introducción a la segunda parte del Manual de Javascript

En esta segunda parte partimos de la base que las personas conocen el lenguaje y la sintaxis y vamos a ver cómo utilizarlo para hacer programación de páginas enriquecidas del lado del cliente.

Introducción al manual de desarrollo del lado del cliente con Javascript

Qué vamos a ver en el manual de programación de Javascript en el cliente web (navegador). Qué contenidos encontrarás en el manual y qué objetivos nos proponemos.

En esta segunda parte del manual de Javascript vamos a tratar de explicar todos los recursos con los que cuenta un programador de Javascript y con los que puede crear todo tipo de efectos y aplicaciones.

Para leer y entender bien lo que viene en los siguientes capítulos es necesario haber leído antes la primera parte de este manual: Programación en Javascript, donde se explican las bases sobre las que tenemos que asentar los siguientes conocimientos, que son básicamente conocer el lenguaje en sí. En la primera parte de este manual conocimos los orígenes y las aplicaciones de Javascript, pero sobretodo hicimos hincapié en su sintaxis, muy importante para entender los scripts que haremos en los siguientes capítulos.

Cuáles son nuestros objetivos en este manual de Javascript para la web

Los objetivos de los siguientes capítulos cubrirán aspectos diversos de Javascript dentro del navegador, lo que hoy conocemos como desarrollo frontend. En resumen serán asuntos como:

- Jerarquía de objetos del navegador (DOM)
- Trabajo con formularios

- Control de ventanas secundarias y frames
- Eventos

Como se puede ver, todos los temas tienen un fuerte carácter práctico y cubren aspectos varios con los que formarnos a nivel superior en Javascript. Ya no es tanto conocer el lenguaje, sino cómo lo aplicaremos en el desarrollo de páginas web.

Por supuesto, todos los artículos serán bastante prácticos e intentaremos que tengan una aplicabilidad directa en el día a día del desarrollo de sitios web. Por ello esperamos que sirvan para iluminar un área tan amplia del desarrollo de páginas web como es el scripting del lado del cliente.

Contenidos adicionales de Javascript

Este manual te ofrecerá un conocimiento ideal para realizar por tu cuenta pequeños comportamientos en páginas web y responder a la interacción del usuario con la página web. Es un pilar fundamental en el aprendizaje de cualquier persona que desee desarrollar proyectos para la web, sin embargo, queremos decir que es solo el inicio de un largo camino para dominar todas las facetas de esta herramienta, ya que a partir de aquí tienes todavía todo un mundo de utilidades para la realización de proyectos.

Existen librerías como jQuery o React, frameworks como Angular o incluso APIs del navegador para hacer cosas increíbles como los Web Components. Por supuesto, para aprender todo esto es importante tener un buen conocimiento de Javascript y de la plataforma web, que vas a adquirir ahora.

Cuando termines este manual estarás en el momento ideal de conocer todas esas otras tecnologías, de las cuales por supuesto tenemos otros manuales dentro de DesarrolloWeb.com. Tienes disponible mucho material dedicado a este lenguaje, ya que es una de las herramientas más importantes en el desarrollo para la web y, actualmente gracias a NodeJS en el desarrollo backend.

No queremos abrumarte con todo el contenido que podrás encontrar más tarde, pero sí dejarte el enlace para encontrarlo todo, desde la sección Javascript.

Vamos sin más pausa con esta segunda parte del manual, que resultará incluso mucho más entretenida y práctica que la primera.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *11/03/2002*
Disponible online en *https://desarrolloweb.com/articulos/704.php*

Los objetos del navegador: DOM de la página

Comenzamos a trabajar con los objetos que nos sirven para controlar directamente los elementos de la página, los objetos que se generan automáticamente en el navegador al visitar una página. A lo largo de estos artículos trataremos diversos componentes del DOM de Javascript (Modelo de Objetos del Documento).

Jerarquía de objetos del navegador (DOM)

Son los objetos que están disponibles en Javascript para controlar cualquier elemento presente en la página web, se conoce normalmente con las siglas DOM, Document Object Model.

Llegamos al tema más importante para aprender a manejar Javascript y controlar lo que ocurre dentro del navegador con toda la potencia que nos ofrece el lenguaje. Se trata de aprender el DOM (Document Object Model o modelo de objetos del navegador) que nos sirve para acceder a cualquiera de los componentes que hay dentro de una página. Por medio del DOM podremos controlar al navegador en general y a los distintos elementos que se encuentran en la página.

Sin duda, este tema le va a dar mucha vida a nuestros ejemplos, ya que hasta ahora no tenían mucho carácter práctico porque no trabajaban con el navegador y las páginas, que es realmente para lo que está hecho Javascript. De modo que esperamos que a partir de aquí el manual sea más entretenido para todos, porque va a cubrir los aspectos más prácticos.

Cuando se carga una página, el navegador crea una jerarquía de objetos en memoria que sirven para controlar los distintos elementos de dicha página. Con Javascript y la nomenclatura de objetos que hemos aprendido, podemos trabajar con esa jerarquía de objetos, acceder a sus propiedades e invocar sus métodos.

> Nota: A lo largo de este manual usamos el término "Jerarquía de objetos del navegador" cuando sería más correcto hablar simplemente del DOM que es como se conoce más técnicamente. El motivo es que cuando se escribió este texto todavía no era común el término DOM, aunque con el tiempo se ha adoptado esa jerga y es la manera como los desarrolladores conocen normalmente al árbol de elementos de la página que están modelados en objetos a los que podemos acceder para cambiar el estado de un documento HTML.

Cualquier elemento de la página se puede controlar de una manera u otra accediendo a esa jerarquía. Es crucial conocerla bien para poder controlar perfectamente las páginas web con

Javascript o cualquier otro lenguaje de programación del lado del cliente.

Ejemplo de acceso a la jerarquía

Antes de empezar a ver rigurosamente la jerarquía de objetos del navegador, vamos a ver el ejemplo típico de acceso a una propiedad de esta jerarquía para cambiar el aspecto de la página. Se trata de una propiedad de la página que almacena el color de fondo de la página web: la propiedad bgColor del objeto document.

```
document.bgColor = "red"
```

Si ejecutamos esta sentencia en cualquier momento cambiamos el color de fondo de la página a rojo. Hay que fijarse en que la propiedad bgColor tiene la "C" en mayúscula. Es un error típico equivocarse con las mayúsculas y minúsculas en la jerarquía. Si no lo escribimos bien no funcionará y en algunos casos ni siquiera dará un mensaje de error.

En esta página definida con color de fondo blanco hemos cambiado esa propiedad luego con Javascript, por lo que saldrá con color de fondo rojo.

```
<HTML>
<HEAD>
    <TITLE>Prueba bgColor</TITLE>
</HEAD>
<BODY bgcolor=white>

<script>
  document.bgColor = "red"
</script>
</BODY>
</HTML>
```

Podemos ver esta página en marcha en una ventana a parte.

En los ejemplos que hemos visto hasta ahora también hemos hecho uso de los objetos de la jerarquía del navegador. En concreto hemos utilizado mucho el método write() del objeto document para escribir un texto en la página.

```
document.write("El texto a escribir")
```

Trabajando con la Jerarquía de objetos del navegador

Vamos a ver ahora como está compuesta esta jerarquía de objetos del navegador, más conocida como DOM, detallando alguno de sus elementos y una explicación sobre como se accede a ellos. Como una imagen vale más que mil palabras, echa un vistazo al gráfico siguiente que contiene un listado parcial de objetos que pueden formar parte de ella.

| | | Jerarquía de objetos del navegador en Javascript 1.2. Podría faltar por recoger algún objeto, pero sirve perfectamente para hacerse una idea de cómo se organizan los objetos en la jerarquía. |

Como se puede apreciar, todos los objetos comienzan en un objeto que se llama window. Este objeto ofrece una serie de métodos y propiedades para controlar la ventana del navegador. Con ellos podemos controlar el aspecto de la ventana, la barra de estado, abrir ventanas secundarias y otras cosas que veremos más adelante cuando expliquemos con detalle el objeto.

Además de ofrecer control, el objeto window da acceso a otros objetos como el documento (La página web que se está visualizando), el historial de páginas visitadas o los distintos frames de la ventana. De modo que para acceder a cualquier otro objeto de la jerarquía deberíamos empezar por el objeto window. Tanto es así que javascript entiende perfectamente que la jerarquía empieza en window aunque no lo señalemos.

En los ejemplos incluidos en el capítulo anterior podíamos haber escrito también las sentencias de acceso a la jerarquía empezando por el objeto window, de esta manera.

```
window.document.bgColor = "red"
window.document.write("El texto a escribir")
```

No lo hicimos por que quedase más claro el código y ahorrar algo de texto, pero ahora ya sabemos que toda la jerarquía empieza en el objeto window.

Las propiedades de un objeto pueden ser a su vez otros objetos

Muchas de las propiedades del objeto window son a su vez otros objetos. Es el caso de objetos como el historial de páginas web o el objeto documento, que tienen a su vez otras propiedades y métodos.

Entre ellos destaca el objeto document, que contiene todas las propiedades y métodos necesarios para controlar muchos aspectos de la página. Ya hemos visto alguna propiedad como bgColor, pero tiene muchas otras como el título de la página, las imágenes que hay incluidas, los formularios, etc. Muchas propiedades de este objeto son a su vez otros objetos, como los formularios. Los veremos todos cuando tratemos cada uno de los objetos por separado. Además, el objeto document tiene métodos para escribir en la página web y para manejar eventos de la página.

Navegación a través de la jerarquía

El objetivo de este capítulo sobre la jerarquía de objetos es aprender a navegar por ella para acceder a cualquier elemento de la página. Esta no es una tarea difícil, pero puede haber algún caso especial en el que acceder a los elementos de la página se haga de una manera que aun no hemos comentado.

Como ya dijimos, toda la jeraquía empieza en el objeto window, aunque no era necesario hacer referencia a window para acceder a cualquier objeto de la jerarquía. Luego en importancia está el objeto document, donde podemos encontrar alguna propiedad especial que merece la pena comentar por separado, porque su acceso es un poco diferente. Se trata de las propiedades que son arrays, por ejemplo la propiedad images es un array con todas las imágenes de la página web. También encontramos arrays para guardar los enlaces de la página, los applets, los formularios y las anclas.

Cuando una página se carga, el navegador construye en memoria la jerarquía de objetos. De manera adicional, construye también estos arrays de objetos. Por ejemplo, en el caso de las imágenes, va creando el array colocando en la posición 0 la primera imagen, en la posición 1 la segunda imagen y así hasta que las introduce todas. Vamos a ver un bucle que recorre todas las

imágenes de la página e imprime su propiedad src, que contiene la URL donde está situada la imagen.

```
for (i=0;i<document.images.length;i++){
	document.write(document.images[i].src)
	document.write("<br>")
}
```

Utilizamos la propiedad length del array images para limitar el número de iteraciones del bucle. Luego utilizamos el método write() del objeto document pasándole el valor de cada una de las propiedades src de cada imagen.

Podemos ver una página con varias imágenes donde se accede a sus propiedades con el bucle anterior.

Ahora vamos a ver el uso de otro array de objetos. En este caso vamos a acceder un poco más dentro de la jerarquía para llegar a la matriz elements de los objetos formulario, que contiene cada uno de los elementos que componen el formulario. Para ello tendremos que acceder a un formulario de la página, al que podremos acceder por el array de formularios, y dentro de él a la propiedad elements, que es otro array de objetos. Para cada elemento del formulario vamos a escribir su propiedad value, que corresponde con la propiedad value que colocamos en HTML.

```
for (i=0;i<document.forms[0].elements.length;i++){
	document.write(document. forms[0].elements[i].value)
	document.write("<br>")
}
```

Es un bucle muy parecido al que teníamos para recorrer las imágenes, con la diferencia que ahora recorremos el vector de elements, al que accedemos por la jerarquía de objetos pasando por el array de formularios en su posición 0, que corresponde con el primer formulario de la página.

Para ver este ejemplo en funcionamiento, tenemos una página con un formulario donde se ejecuta este recorrido a sus elementos.

Con esto hemos aprendido a movernos por la jerarquía de objetos, con lo que podremos acceder a cualquier elemento del navegador o la página. En adelante conoceremos con detalle cada uno de los objetos de la jerarquía, empezando por el objeto window y bajando por la jerarquía hasta verlos todos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *11/06/2002*
Disponible online en *https://desarrolloweb.com/articulos/807.php*

Objeto window de Javascript

Estudiamos el objeto window de Javascript que nos sirve para controlar la ventana del navegador. Detallamos sus propiedades y hacemos un ejemplo.

Es el objeto principal en la jerarquía y contiene las propiedades y métodos para controlar la ventana del navegador. De él dependen todos los demás objetos de la jerarquía. Vamos a ver la lista de sus propiedades y métodos.

Propiedades del objeto window

A continuación podemos ver las propiedades del objeto window. Hay algunas muy útiles y otras que lo son menos.

closed

Indica la posibilidad de que se haya cerrado la ventana. (Javascript 1.1)

defaultStatus

Texto que se escribe por defecto en la barra de estado del navegador.

document

Objeto que contiene el la página web que se está mostrando.

Frame

Un objeto frame de una página web. Se accede por su nombre.

frames array

El vector que contiene todos los frames de la página. Se accede por su índice a partir de 0.

history

Objeto historial de páginas visitadas.

innerHeight

Tamaño en pixels del espacio donde se visualiza la página, en vertical. (Javascript 1.2)

innerWidth

Tamaño en pixels del espacio donde se visualiza la página, en horizontal. (Javascript 1.2)

length

Numero de frames de la ventana.

location

La URL del documento que se está visualizando. Podemos cambiar el valor de esta propiedad para movernos a otra página. Ver también la propiedad location del objeto document.

locationbar

Objeto barra de direcciones de la ventana. (Javascript 1.2)

menubar

Objeto barra de menús de la ventana. (Javascript 1.2)

name

Nombre de la ventana. Lo asignamos cuando abrimos una nueva ventana.

opener

Hace referencia a la ventana de navegador que abrió la ventana donde estamos trabajando. Se verá con detenimiento en el tratamiento de ventanas con Javascript.

outherHeight

Tamaño en pixels del espacio de toda la ventana, en vertical. Esto incluye las barras de desplazamiento, botones, etc. (Javascript 1.2)

outherWidth

Tamaño en pixels del espacio de toda la ventana, en horizontal. Esto incluye las barras de desplazamiento. (Javascript 1.2)

parent

Hace referencia a la ventana donde está situada el frame donde estamos trabajando. La veremos con detenimiento al estudiar el control de frames con Javascript.

personalbar

Objeto barra personal del navegador. (Javascript 1.2)

self

Ventana o frame actual.

scrollbars

Objeto de las barras de desplazamiento de la ventana.

status

Texto de la barra de estado.

statusbar

Objeto barra de estado del navegador. (Javascript 1.2)

toolbar

Objeto barra de herramientas. (Javascript 1.2)

top

Hace referencia a la ventana donde está situada el frame donde estamos trabajando. Como la propiedad parent.

window

Hace referencia a la ventana actual, igual que la propiedad self.

Vamos a ver un ejemplo de utilización de la propiedad status del objeto window. Esta propiedad sirve para escribir un texto en la barra de estado del navegador (la barra de debajo de la ventana). En este ejemplo hemos tenido que adelantarnos un poco en la marcha del manual, pues utilizamos un manejador de eventos y no hemos visto todavía lo que son. En concreto utilizamos el manejador de eventos onclick, que sirve para ejecutar sentencias Javascript cuando el usuario pulsa un elemento de la página.

Los manejadores de eventos se colocan en etiquetas HTML, en nuestro caso lo colocamos en un botón de formulario. Las sentencias Javascript asociadas al evento onclick del botón se ejecutarán cuando pulsemos el botón.

Veamos ya el código que hace que se cambie el texto de la barra de estado cuando pulsemos un botón.

```
<form>
<input type="Button" value="Pulsame!" onclick="window.status='Hola a todo el mundo!'">
</form>
```

Simplemente asignamos un texto a la propiedad status del objeto window. El texto que

colocamos en la barra de estado está escrito entre comillas simples porque estamos escribiendo dentro de unas comillas dobles.

Podemos ver una página a parte con este ejemplo.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *02/07/2002*
Disponible online en *https://desarrolloweb.com/articulos/826.php*

Métodos de window en Javascript

El objeto window de Javascript tiene a disposición de los programadores una larga lista de métodos. Los estudiamos y vemos ejemplos.

Vamos a ver ahora los distintos métodos que tiene el objeto window. Muchos de estos métodos habrá que verlos por separado porque son muy útiles y aun no los hemos utilizado, ahora vamos a listarlos y ya veremos algunos ejemplos.

alert(texto)

Presenta una ventana de alerta donde se puede leer el texto que recibe por parámetro

back()

Ir una página atrás en el historial de páginas visitadas. Funciona como el botón de volver de la barra de herramientas. (Javascript 1.2)

blur()

Quitar el foco de la ventana actual. (Javascript 1.1)

captureEvents(eventos)

Captura los eventos que se indiquen por parámetro (Javascript 1.2).

clearInterval()

Elimina la ejecución de sentencias asociadas a un intervalo indicadas con el método setInterval().(Javascript 1.2)

clearTimeout()

Elimina la ejecución de sentencias asociadas a un tiempo de espera indicadas con el método

setTimeout().

close()

Cierra la ventana. (Javascript 1.1)

confirm(texto)

Muestra una ventana de confirmación y permite aceptar o rechazar.

find()

Muestra una ventanita de búsqueda. (Javascript 1.2 para Netscape)

focus()

Coloca el foco de la aplicación en la ventana. (Javascript 1.1)

forward()

Ir una página adelante en el historial de páginas visitadas. Como si pulsásemos el botón de adelante del navegador. (Javascript 1.2)

home()

Ir a la página de inicio que haya configurada en el explorador. (Javascript 1.2)

moveBy(pixelsX, pixelsY)

Mueve la ventana del navegador los pixels que se indican por parámetro hacia la derecha y abajo. (Javascript 1.2)

moveTo(pixelsX, pixelsY)

Mueve la ventana del navegador a la posición indicada en las coordenadas que recibe por parámetro. (Javascript 1.2)

open()

Abre una ventana secundaria del navegador. Se puede aprender a utilizarla en el reportaje de cómo abrir ventanas secundarias.

print()

Como si pulsásemos el botón de imprimir del navegador. (Javascript 1.2)

prompt(pregunta,inicializacion_de_la_respuesta)

Muestra una caja de diálogo para pedir un dato. Devuelve el dato que se ha escrito.

releaseEvents(eventos)

Deja de capturar eventos del tipo que se indique por parámetro. (Javascript 1.2)

resizeBy(pixelsAncho,pixelsAlto)

Redimensiona el tamaño de la ventana, añadiendo a su tamaño actual los valores indicados en los parámetros. El primero para la altura y el segundo para la anchura. Admite valores negativos si se desea reducir la ventana. (Javascript 1.2)

resizeTo(pixelsAncho,pixelsAlto)

Redimensiona la ventana del navegador para que ocupe el espacio en pixels que se indica por parámetro (Javascript 1.2)

routeEvent()

Enruta un evento por la jerarquía de eventos. (Javascript 1.2)

scroll(pixelsX,pixelsY)

Hace un scroll de la ventana hacia la coordenada indicada por parámetro. Este método está desaconsejado, pues ahora se debería utilizar scrollTo()(Javascript 1.1)

scrollBy(pixelsX,pixelsY)

Hace un scroll del contenido de la ventana relativo a la posición actual. (Javascript 1.2)

scrollTo(pixelsX,pixelsY)

Hace un scroll de la ventana a la posición indicada por el parámetro. Este método se tiene que utilizar en lugar de scroll. (Javascript 1.2)

setInterval()

Define un script para que sea ejecutado indefinidamente en cada intervalo de tiempo. (Javascript 1.2)

setTimeout(sentencia,milisegundos)

Define un script para que sea ejecutado una vez después de un tiempo de espera determinado.

stop()

Como pulsar el botón de stop de la ventana del navegador. (Javascript 1.2)

Para ilustrar un poco mejor el funcionamiento de alguno de estos métodos -los más extraños-, hemos creado una página web que los utiliza. El código de la página se muestra a continuación y también podemos ver la página en marcha.

```
<form>
<input type="button" value="Ventana de búsqueda (Solo Netscape)" onClick="window.find()">
<br>
<br>
<input type="button" value="Mover la ventana 10 derecha,10 abajo" onClick="moveBy(10, 10)">
<br>
<br>
<input type="button" value="Mover la ventana al punto (100,10)" onClick="moveTo(100, 10)">
<br>
<br>
<input type="button" value="Imprimir esta página" onClick="print()">
<br>
<br>
<input type="button" value="Aumenta la ventana 10 ancho,10 largo" onClick="resizeBy(10, 10)">
<br>
<br>
<input type="button" value="Fija el tamaño de la ventana en 400 x 200" onClick="resizeTo(400, 200)">
<br>
<br>
<input type="button" value="Scroll arriba del todo" onClick="scroll(0,0)">
<br>
<br>
<input type="button" value="Scroll arriba 10 pixels" onClick="scrollBy(0,-10)">
</form>
```

Estos ejemplos son muy simples, aunque poco prácticos. Únicamente se trata de una serie de botones que, al pulsarlos, llaman a otros tantos métodos del objeto window. En el atributo onclick de la etiqueta del botón se indican las sentencias Javascript que queremos que se ejecuten cuando se pulsa sobre dicho botón.

En el capítulo siguiente veremos otros ejemplos realizados con métodos del objeto window de Javascript, un poco más detallados.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *02/07/2002*
Disponible online en *https://desarrolloweb.com/articulos/827.php*

Ejemplos de métodos de window

Otros ejemplos de métodos del objeto window de Javascript, relatados con detalle.

Ahora vamos a realizar algún ejemplo de utilización de los métodos de la ventana. Nos vamos a

centrar en los ejemplos que sirven para sacar cajas de diálogo, que son muy útiles.

Caja de alerta

Para sacar un texto en una ventanita con un botón de aceptar. Recibe el texto por parámetro.

```
window.alert("Este es el texto que sale")
```

Como el objeto window se sobreentiende podemos escribirlo así.

```
alert("Este es el texto que sale")
```

Saca una ventana como la que se puede ver en esta página.

Caja de confirmación

Muestra una ventana con un texto indicado por parámetro y un botón de aceptar y otro de rechazar. Dependiendo del botón que se pulsa devuelve un true (si se pulsa aceptar) o un false (si se pulsa rechazar).

```
<script>
var respuesta = confirm("Aceptame o rechazame")
alert ("Has pulsado: " + respuesta)
</script>
```

Este script muestra una caja de diálogo confirm y luego muestra en otra caja de diálogo alert el contenido de la variable que devolvió la caja de diálogo. Nuevamente, podemos ver el funcionamiento de este script si accedemos a esta página a parte.

Caja de introducción de un dato

Muestra una caja de diálogo donde se formula una pregunta y hay un campo de texto para escribir una respuesta. El campo de texto aparece relleno con lo que se escriba en el segundo parámetro del método. También hay un botón de aceptar y otro de cancelar. En caso de pulsar aceptar, el método devuelve el texto que se haya escrito. Si se pulsó cancelar devuelve null.

El ejemplo siguiente sirve para pedir el nombre de la persona que está visitando la página y luego mostrar en la página un saludo personalizado. Utiliza un bucle para repetir la toma de datos siempre que el nombre de la persona sea null (porque pulsó el botón de cancelar) o sea un string vacío (porque no escribió nada).

```
<script>
nombre = null
while (nombre == null || nombre == ""){
      nombre = prompt("Dime tu nombre:","")
}
document.write("<h1>Hola " + nombre + "</h1>")
</script>
```

Si nos fijamos en la caja prompt veremos que recibe dos parámetros. El segundo era el texto por defecto que sale en la caja como respuesta. Lo hemos dejado como un string vacío para que no salga nada como texto por defecto.

Podemos ver este último script en funcionamiento en una página a parte.

Hasta aquí los ejemplos de los métodos del objeto window. De todos modos, en el resto del manual tendremos ocasión de ver cómo trabajar con muchas propiedades y métodos de este objeto.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *02/07/2002*
Disponible online en *https://desarrolloweb.com/articulos/828.php*

Objeto document en Javascript

Una descripción del objeto de Javascript que sirve para controlar el domumento que se visualiza en el navegador. También hay una lista de todas sus propiedades.

Con el objeto document se controla la página web y todos los elementos que contiene. El objeto document es la página actual que se está visualizando en ese momento. Depende del objeto window, pero también puede depender del objeto frame en caso de que la página se está mostrando en un frame.

Propiedades del objeto document

Veamos una lista de las propiedades del objeto document y luego veremos algún ejemplo.

alinkColor

Color de los enlaces activos

Anchor

Un ancla de la página. Se consigue con la etiqueta <A name="nombre_del_ancla">. Se accede por su nombre.

anchors array

Un array de las anclas del documento.

Applet

Un applet de la página. Se accede por su nombre. (Javascript 1.1)

applets array

Un array con todos los applets de la página. (Javascript 1.1)

Area

Una etiqueta <AREA>, de las que están vinculadas a los mapas de imágenes (Etiqueta MAP). Se accede por su nombre. (Javascript 1.1)

bgColor

El color de fondo del documento.

classes

Las clases definidas en la declaración de estilos CSS. (Javascript 1.2)

cookie

Una cookie

domain

Nombre del dominio del servidor de la página.

Embed

Un elemento de la pagina incrustado con la etiqueta <EMBED>. Se accede por su nombre. (Javascript 1.1)

embeds array

Todos los elementos de la página incrustados con <EMBED>. (Javascript 1.1)

fgColor

El color del texto. Para ver los cambios hay que reescribir la página.

From

Un formulario de la página. Se accede por su nombre.

forms array

Un array con todos los formularios de la página.

ids

Para acceder a estilos CSS. (Javascript 1.2)

Image

Una imagen de la página web. Se accede por su nombre. (Javascript 1.1)

images array

Cada una de las imágenes de la página introducidas en un array. (Javascript 1.1)

lastModified

La fecha de última modificación del documento.

linkColor

El color de los enlaces.

Link

Un enlace de los de la página. Se accede por su nombre.

links array

Un array con cada uno de los enlaces de la página.

location

La URL del documento que se está visualizando. Es de solo lectura.

referrer

La página de la que viene el usuario.

tags

Estilos definidos a las etiquetas de HTML en la página web. (Javascript 1.2)

title

El titulo de la página.

URL

Lo mismo que location, pero es aconsejable utilizar location ya que URL no existe en todos los navegadores.

vlinkColor

El color de los enlaces visitados.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *22/07/2002*
Disponible online en *https://desarrolloweb.com/articulos/846.php*

Ejemplos de propiedades de document

Vemos varios ejemplos de acceso y manipulación de las propiedades del objeto document de Javascript.

Después de estudiar las propiedades del objeto document, vamos a ver algún ejemplo para ilustrar el modo de acceso y utilización de las mismas.

Ejemplo con la propiedad bgColor

Vamos a utilizar el evento onclick para colocar tres botones en la página que al pulsarlos nos cambie el color del fondo de la página.

```
<script>
function cambiaColor(colorin){
	document.bgColor = colorin
}

</script>
<form>
<input type="Button" value="Rojo" onclick="cambiaColor('ff0000')">
<input type="Button" value="Verde" onclick="cambiaColor('00ff00')">
<input type="Button" value="Azul" onclick="cambiaColor('0000ff')">
</form>
```

Primero definimos una función que será la encargada de cambiar el color y luego tres botones que llamarán a la función cuando sean pulsados pasándole el color como parámetro.

Podemos ver el ejemplo en marcha.

Propiedad title

La propiedad title guarda la cadena que conforma el título de nuestra página web. Si accedemos a dicha propiedad obtendremos el título y si la cambiamos, cambiará el título de la página web.

> Nota: recordamos que el título se puede ver en la barra de arriba del todo de la ventana del navegador.

Vamos a mostrar e título de la página en una caja de alerta.

```
alert (document.title)
```

Ahora vamos a hacer una función que modifica el título de la página , asignándole el texto que le llegue por parámetro.

```
function cambiaTitulo(texto){
    document.title = texto
}
```

Como en el ejemplo anterior, vamos a crear varios botones que llamen a la función pasándole distintos textos, que se colocarán en el título de la página.

```
<form>
<input type="Button" value="Titulo = Hola a todos" onclick="cambiaTitulo('Hola a todos')">
<input type="Button" value="Titulo = Bienvenidos a mi página web" onclick="cambiaTitulo('Bienvenidos a mi página web')">
<input type="Button" value="Titulo = Más dias que longanizas" onclick="cambiaTitulo('Más dias que longanizas')">
</form>
```

Podemos ver en funcionamiento este script en otra página web.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *22/07/2002*
Disponible online en *https://desarrolloweb.com/articulos/847.php*

Métodos de document

Vemos una lista de los eventos disponibles en el objeto document.

El objeto document, localizado debajo del objeto window en la jerarquía de objetos de Javascript, también tiene una lista de métodos interesantes. La vemos a continuación.

captureEvents()

Para capturar los eventos que ocurran en la página web. Recibe como parámetro el evento que se desea capturar.

close()

Cierra el flujo del documento. (Se verá más adelante en este manual un artículo sobre el flujo del documento)

contextual()

Ofrece una línea de control de los estilos de la página. En el caso que deseemos especificarlos con Javascript.

getSelection()

Devuelve un string que contiene el texto que se ha seleccionado. Sólo funciona en Netscape.

handleEvent()

Invocas el manejador de eventos del elemento especificado.

open()

Abre el flujo del documento.

releaseEvents()

Liberar los eventos capturados del tipo que se especifique, enviándolos a los objetos siguientes en la jerarquía.

routeEvent()

Pasa un evento capturado a través de la jerarquía de eventos habitual.

write()

Escribe dentro de la página web. Podemos escribir etiquetas HTML y texto normal.

writeln()

Escribe igual que el método write(), aunque coloca un salto de línea al final.

Los eventos de document sirven principalmente para controlar dos cosas. Un grupo nos ofrece

una serie de funciones para el control de los documentos, como escribir, abrirlos y cerrarlos. Los veremos en el capítulo siguiente que habla sobre el control del flujo de escritura del documento. El otro grupo ofrece herramientas para el control de los eventos en el documento y lo veremos más adelante cuando tratemos con detenimiento el tema de eventos.

Se nos queda un poco suelto el método getSelection(), que sólo funciona en los navegadores de Netscape y, por tanto, no resulta muy útil para aplicaciones que deseemos que sean compatibles en todos los sistemas. Aun así, haremos el ejemplo sobre este método, ya que los otros los vamos a ver en siguientes capítulos.

El ejemplo consiste en una página que tiene un poco de texto y un botón. En la página podremos seleccionar algo de texto y luego apretar el botón, que llama a una función que muestra en una caja alert el texto que se ha seleccionado. El código es el siguiente:

```
<html>
<head>
<title>Rescatar lo seleccionado</title>
<script language="JavaScript">
function mostrarSeleccionado(){
      alert("Has seleccionado:n" + document.getSelection())
}
</script>
</head>

<body>
<h1>Rescatar lo seleccionado</h1>
<br>

<form>
<input type="button" value="pulsame!" onClick="mostrarSeleccionado()">
</form>

Selecciona cualquier texto de la página y pulsa sobre el botón.

</body>
</html>
```

Se puede ver en funcionamiento el script en una página aparte, aunque sólo funcionará en Netscape e Internet Explorer dará un error.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *12/08/2002*
Disponible online en *https://desarrolloweb.com/articulos/861.php*

Flujo de escritura del documento

Es el proceso en el que el navegador escribe la página. Para escribir en la página desde Javascript el flujo del documento debe estar abierto.

Acerca del objeto document también es interesante hablar un poco sobre el flujo de escritura del documento o página web.

Cuando el navegador lee una página web la va interpretando y escribiendo en su ventana. El proceso en el que el navegador está escribiendo la página le llamamos flujo de escritura del documento. El flujo comienza cuando se empieza a escribir la página y dura hasta que ésta ha terminado de escribirse. Una vez terminada la escritura de la página el flujo de escritura del documento se cierra automáticamente. Con esto termina la carga de la página.

Una vez cerrado el flujo no se puede escribir en la página web, ni texto ni imágenes ni otros elementos.

En javascript tenemos que respetar el flujo de escritura del documento forzosamente. Es por ello que sólo podemos ejecutar sentencias document.write() (método write() del objeto document) cuando está abierto el flujo de escritura del documento, o lo que es lo mismo, mientras se está cargando la página.

Si recordamos las dos formas de ejecutar un script en Javascript:

- Ejecución de los scripts mientras que carga la página. Aquí podremos ejecutar document.write() y lo hemos hecho habitualmente en los ejemplos anteriores.
- Ejecución de los scripts cuando la página ha sido cargada, como respuesta a un evento del usuario. Aquí no podemos hacerlo porque la página ha terminado de cargarse, de hecho, no lo hemos hecho nunca hasta ahora.

Hay un matiz que dar a que no se puede escribir en la página cuando ya está cerrado el flujo. En realidad si que se puede, pero necesitamos abrir el flujo otra vez para escribir en la página, tanto es así que aunque nosotros no lo abramos explícitamente Javascript se encargará de ello. Lo que tiene que quedar claro es que si hacemos un document.write() el flujo tiene que estar abierto y si no lo está se abrirá. El problema de abrir el flujo de escritura del documento una vez ya está escrita la página es que se borra todo su contenido.

Para que quede claro vamos a hacer un script para escribir en la página una vez ésta ha sido cargada. Simplemente necesitamos un botón y al pulsarlo ejecutar el método write() del objeto document.

```
<form>
<INPUT type=button value=escribir onclick="window.document.write('hola')">
</form>
```

Si nos fijamos, en el manejador de eventos onclick hemos colocado la jerarquía de objetos desde el principio, es decir, empezando por el objeto window. Esto es porque hay algunos navegadores que no sobreentienden el objeto window en las sentencias escritas en los manejadores de eventos.

Podemos ver el ejemplo en funcionamiento.

El resultado de la ejecución puede variar de un navegador a otro, pero en cualquier caso se borrará la página que se está ejecutando.

Métodos open() y close() de document

Los métodos open() y close() del objeto document sirven para controlar el flujo del documento desde Javascript. Nos permiten abrir y cerrar el documento explícitamente.

El ejemplo de escritura en la página anterior se debería haber escrito con su correspondiente apertura y cierre del documento y hubiese quedado algo parecido a esto.

```
<script>
function escribe(){
      document.open()
      window.document.write('Hola')
      document.close()
}
</script>
<form>
<INPUT type=button value=escribir onclick="escribe()">
</form>
```

Vemos que ahora no escribimos las sentencias dentro del manejador, porque, cuando hay más de una sentencia, queda más claro poner una llamada a una función y en la función colocamos las sentencias que queramos.

Las sentencias del ejemplo anterior, que cubren la apertura, escritura y cierre del documento. Se pueden ver en marcha aquí.

Trabajo con formularios en Javascript

Los formularios forman parte del DOM de la página y son un grupo de elementos con los que trabajaremos muy habitualmente y que por tanto, merece la pena prestar especial atención. Aprenderemos a recibir y alterar dinámicamente los valores y estados de los elementos de formulario con Javascript.

Trabajo con formularios en Javascript

Para continuar vamos a ver una serie de capítulos enfocados a aprender a trabajar con los formularios. Ahora veremos como acceder a los formularios y sus elementos.

Para continuar vamos a ver una serie de capítulos enfocados a aprender a trabajar con los formularios, acceder a sus elementos para controlar sus valores y actualizarlos.

El objeto form depende en la jerarquía de objetos del objeto document. En un objeto form podemos encontrar algunos métodos y propiedades, pero lo más destacado que podremos encontrar son cada uno de los elementos del formulario. Es decir, de un formulario dependen todos los elementos que hay dentro, como pueden ser campos de texto, cajas de selección, áreas de texto, botones de radio, etc.

Para acceder a un formulario desde el objeto document podemos hacerlo de dos formas.

1. A partir de su nombre, asignado con el atributo NAME de HTML.
2. Mediante la matriz de formularios del objeto document, con el índice del formulario al que queremos acceder.

Para este formulario

```
<FORM name="f1">
<INPUT type=text name=campo1>
<INPUT type=text name=campo2>
</FORM>
```

Podremos acceder con su nombre de esta manera.

```
document.f1
```

O con su índice, si suponemos que es el primero de la página.

```
document.forms[0]
```

De similar manera accedemos a los elementos de un formulario, que dependen del objeto form.

1. A partir del nombre del objeto asignado con el atributo NAME de HTML.
2. Mediante la matriz de elementos del objeto form, con el índice del elemento al que queremos acceder.

Podríamos acceder al campo 1 del anterior formulario de dos maneras. Con su nombre lo haríamos así.

```
document.f1.campo1
```

o a partir del array de elementos.

document.f1.elements[0] (utilizamos el índice 0 porque es el primer elemento y en Javascript los arrays empiezan por 0.)

Si deseamos acceder al segundo campo del formulario escribiríamos una de estas dos cosas:

```
document.f1.campo2
document.f1.elements[1]
```

recordamos que también podemos acceder a un fomulario por el array de forms, indicando el índice del formulario al que acceder. De este modo, el acceso al campo2 sería el siguiente:

```
document.forms[0].campo2
document.forms[0].elements[1]
```

En estos casos hemos supuesto que este formulario es el primero que hay escrito en el código HTML, por eso accedemos a él con el índice 0.

Esperamos que haya quedado claro el acceso a formularios y sus campos. Pasaremos entonces, sin más, a un ejemplo para practicar todo esto.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *27/09/2002*
Disponible online en *https://desarrolloweb.com/articulos/910.php*

Ej. trabajo con formularios. Calculadora sencilla

Vamos a ver un ejemplo del trabajo con formularios en el que desarrollaremos una calculadora sencilla.

Para ilustrar un poco el trabajo con formularios, vamos a realizar un ejemplo práctico. Puede

que algunas cosas que vamos a tratar queden un poco en el aire porque no se hayan explicado con detenimiento antes, pero seguro que nos sirve para enterarnos de cómo se trabaja con formularios y las posibilidadedes que tenemos.

Ejemplo calculadora sencilla

En este ejemplo vamos a construir una calculadora, aunque bastante sencilla, que permita realizar las operaciones básicas. Para hacer la calculadora vamos a realizar un formulario en el que vamos a colocar tres campos de texto, los dos primeros para los operandos y un tercero para el resultado. Además habrán unos botones para hacer las operaciones básicas.

El formulario de la calculadora se puede ver aquí.

```
<form name="calc">
<input type="Text" name="operando1" value="0" size="12">
<br>
<input type="Text" name="operando2" value="0" size="12">
<br>
<input type="Button" name="" value=" + " onclick="calcula('+')">
<input type="Button" name="" value=" - " onclick="calcula('-')">
<input type="Button" name="" value=" X " onclick="calcula('*')">
<input type="Button" name="" value=" / " onclick="calcula('/')">
<br>
<input type="Text" name="resultado" value="0" size="12">
</form>
```

Mediante una función vamos a acceder a los campos del formulario para recoger los operandos en dos variables. Los campos de texto tienen una propiedad llamada value que es donde podemos obtener lo que tienen escrito en ese momento. Mas tarde nos ayudaremos de la función eval() para realizar la operación. Pondremos por último el resultado en el campo de texto creado en tercer lugar, utilizando también la propiedad value del campo de texto.

A la función que realiza el cálculo (que podemos ver a continuación) la llamamos apretando los botones de cada una de las operaciones. Dichos botones se pueden ver en el formulario y contienen un atributo onclick que sirve para especificar las sentencias Javascript que deseamos que se ejecuten cuando el usuario pulse sobre él. En este caso, la sentencia a ejecutar es una llamada a la función calcula() pasando como parámetro el símbolo u operador de la operación que deseamos realizar.

Script con la función calcula()

```
<script>
function calcula(operacion){
    var operando1 = document.calc.operando1.value
    var operando2 = document.calc.operando2.value
    var result = eval(operando1 + operacion + operando2)
    document.calc.resultado.value = result
}
</script>
```

La función eval(), recordamos, que recibía un string y lo ejecutaba como una sentencia Javascript. En este caso va a recibir un número que concatenado a una operación y otro

número será siempre una expresión aritmética que eval() solucionará perfectamente.

Podemos ver el ejemplo de la calculadora en funcionamiento.

El acceso a otros elementos de los formularios se hace de manera parecida en cuanto respecta a la jerarquía de objetos, aunque como cada elemento tiene sus particularidades las cosas que podremos hacer con ellos diferirán un poco. Lo veremos un poco más adelante.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *15/10/2002*
Disponible online en *https://desarrolloweb.com/articulos/929.php*

Propiedades y métodos del objeto form

Echamos un vistazo a las distintas propiedades y métodos del objeto form de Javascript. Mostramos algún ejemplo de utilización de propiedades y una sencilla validación de formulario y envío con el método submit().

Vamos a ver ahora el objeto form por si solo, para destacar sus propiedades y métodos.

Propiedades del objeto form

Tiene unas cuantas propiedades para ajustar sus atributos mediante Javascript.

action

Es la acción que queremos realizar cuando se submite un formulario. Se coloca generalmente una dirección de correo o la URL a la que le mandaremos los datos. Corresponde con el atributo ACTION del formulario.

elements array

La matriz de elementos contiene cada uno de los campos del formulario.

encoding

El tipo de codificación del formulario

length

El número de campos del formulario.

method

El método por el que mandamos la información. Corresponde con el atributo METHOD del formulario.

name

El nombre del formulario, que corresponde con el atributo NAME del formulario.

target

La ventana o frame en la que está dirigido el formulario. Cuando se submita se actualizará la ventana o frame indicado. Corresponde con el atributo target del formulario.

Ejemplos de trabajo con las propiedades

Con estas propiedades podemos cambiar dinámicamente con Javascript los valores de los atributos del formulario para hacer con él lo que se desee dependiendo de las exigencias del momento.

Por ejemplo podríamos cambiar la URL que recibiría la información del formulario con la instrucción.

```
document.miFormulario.action = "miPágina.asp"
```

O cambiar el target para submitir un formulario en una posible ventana secundaria llamada mi_ventana.

```
document.miFormulario.target = "mi_ventana"
```

Métodos del objeto form

Estos son los métodos que podemos invocar con un formulario.

submit()

Para hacer que el formulario se submita, aunque no se haya pulsado el botón de submit.

reset()

Para reinicializar todos los campos del formulario, como si se hubiese pulsado el botón de reset. (Javascript 1.1)

Ejemplo de trabajo con los métodos

Vamos a ver un ejemplo muy sencillo sobre cómo validar un formulario para submitirlo en caso de que esté relleno. Para ello vamos a utilizar el método submit() del formulario.

El mecanismo es el siguiente: en vez de colocar un botón de submit colocamos un botón normal (<INPUT type="button">) y hacemos que al pulsar ese botón se llame a una función que es la encargada de validar el formulario y, en caso de que esté correcto, submitirlo.

El formulario quedaría así.

```
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="button" value="Enviar" onclick="validaSubmite()">
</form>
```

Nos fijamos en que no hay botón de submit, sino un botón normal con una llamada a una función que podemos ver a continuación.

```
function validaSubmite(){
    if (document.miFormulario.campo1.value == "")
      alert("Debe rellenar el formulario")
    else
      document.miFormulario.submit()
}
```

En la función se comprueba si lo que hay escrito en el formulario es un string vacío. Si es así se muestra un mensaje de alerta que informa que se debe rellenar el formulario. En caso de haya algo en el campo de texto submite el formulario utilizando el método submit del objeto form.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *30/10/2002*
Disponible online en *https://desarrolloweb.com/articulos/947.php*

Control de campos de texto con Javascript

Explicación y documentación de los campos de texto y su control con Javascript. Se incluyen los campos de tipo text, password y hidden.

Vamos a ver ahora los campos donde podemos guardar cadenas de texto, es decir, los campos de texto, password y hidden. Hay otro campo relacionado con la escritura de texto, el campo TextArea, que veremos más adelante.

Campo Text

Es el campo que resulta de escribir la etiqueta <INPUT type="text">. Lo hemos utilizado hasta el momento en varios ejemplos, pero vamos a parar un momento en él para describir sus propiedades y métodos.

Propiedades del campo text

Vemos la lista de propiedades de estos tipos de campo.

defaultValue

Es el valor por defecto que tiene un campo. Lo que contiene el atributo VALUE de la etiqueta <INPUT>.

form

Hace referencia al formulario.

name

Contiene el nombre de este campo de formulario

type

Contiene el tipo de campo de formulario que es.

value

El texto que hay escrito en el campo.

Vamos a ver un ejemplo sobre lo que puede hacer la propiedad defaultValue. En este ejemplo tenemos un formulario y un botón de reset. Si pulsamos el botón de reset el campo de texto se vacía porque su value de HTML era un string vacío. Pero si pulsamos el botón siguiente llamamos a una función que cambia el valor por defecto de ese campo de texto, de modo que al pulsar el botón de reset mostrará el nuevo valor por defecto.

Este es el código de la página completa.

```
<html>
<head>
  <title>Cambiar el valor por defecto</title>
  <script>
   function cambiaDefecto(){
        document.miFormulario.campo1.defaultValue = "Hola!!"
   }
  </script>
</head>

<body>
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="Reset">
<br>
<br>
<input type="button" value="Cambia valor por defecto" onclick="cambiaDefecto()">
</form>
</body>
</html>
```

Se puede ver en funcionamiento en esta página.

Métodos del objeto Text

Se pueden invocar los siguientes métodos sobre los objetos tipo Text.

blur()

Retira el foco de la aplicación del campo de texto.

focus()

Pone el foco de la aplicación en el campo de texto.

select()

Selecciona el texto del campo.

Como ejemplo vamos a mostrar una función que selecciona el texto de un campo de texto de un formulario como el de la página del ejemplo anterior. Para hacerlo hemos utilizado dos métodos, el primero para pasar el foco de la aplicación al campo de texto y el segundo para seleccionar el texto.

```
function seleccionaFoco(){
    document.miFormulario.campo1.focus()
    document.miFormulario.campo1.select()
}
```

Puede verse en funcionamiento en esta página.

Campos Password

Estos funcionan igual que los hidden, con la peculiaridad que el contenido del campo no puede verse escrito en el campo, por lo que salen asteríscos en lugar del texto.

Campos Hidden

Los campos hidden son como campos de texto que están ocultos para el usuario, es decir, que no se ven en la página. Son muy útiles en el desarrollo de webs para pasar variables en los formularios a las que no debe tener acceso el usuario.

Se colocan en con HTML con la etiqueta <INPUT type=hidden> y se rellenan de datos con su atributo value. Mas tarde podremos cambiar el dato que figura en el campo accediendo a la propiedad value del campo de texto igual que lo hemos hecho antes.

```
document.miFormulario.CampoHidden.value = "nuevo texto"
```

El campo hidden sólo tiene algunas de las propiedades de los campos de texto. En concreto

tiene la propiedad value y las propiedades que son comunes de todos los campos de formulario: name, from y type, que ya se describieron para los campos de texto.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *18/11/2002*
Disponible online en *https://desarrolloweb.com/articulos/970.php*

Control de Checkbox en Javascript

Cómo realizar el control del elemento de formulario tipo checkbox o caja de verificación. Estudiamos los métodos y propiedades de los elementos checkbox, con algunos ejemplos.

Los checkbox son uno de los elementos de formulario de HTML. Consiste en unas cajas que permiten marcarlas o no para verificar alguna cosa en un formulario. Podemos ver el aspecto de una caja checkbok a continuación.

Los checkbox se consiguen con la etiqueta <INPUT type="checkbox">.

- Con el atributo NAME de la etiqueta INPUT le podemos dar un nombre para luego acceder a ella con javascript.
- Con el atributo CHECKED indicamos que el campo debe aparecer chequeado por defecto.
- El atributo CHECKED es boleano. Si colocamos ese atributo indica que el elemento estará chequeado inicialmente, si el atributo no se encuentra en la etiqueta el elemento aparecerá desmarcado.

Con Javascript, a partir de la jerarquía de objetos del navegador, el DOM, tenemos acceso al checkbox, que depende del objeto form del formulario donde está incluido.

Propiedades de un checkbox

Las propiedades que tiene un checkbox son las siguientes.

checked

Informa sobre el estado del checkbox. Puede ser true o false.

defaultChecked

Si está chequeada por defecto o no.

value

El valor actual del checkbox.

También tiene las propiedades form, name, type como cualquier otro elemento de formulario.

Métodos del checkbox

Veamos la lista de los métodos de un checkbox.

click()

Es como si hiciésemos un click sobre el checkbox, es decir, cambia el estado del checkbox.

blur()

Retira el foco de la aplicación del checkbox.

focus()

Coloca el foco de la aplicación en el checkbox.

Ejemplo práctico del manejo de checkboxes

Para ilustrar el funcionamiento de las casillas checkbox vamos a ver una página que tiene un checkbox y tres botones. Los dos primeros para mostrar las propiedades checked y value y el tercero para invocar a su método click() con objetivo de simular un click sobre el checkbox.

```
<html>
<head>
    <title>Ejemplo Checkbox</title>
<script>
function alertaChecked(){
    alert(document.miFormulario.miCheck.checked);
}
function alertaValue(){
    alert(document.miFormulario.miCheck.value);
}
function metodoClick(){
    document.miFormulario.miCheck.click();
}
</script>
</head>
```

```
<body>
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="checkbox" name="miCheck">
<br>
<br>
<input type="button" value="informa de su propiedad checked" onclick="alertaChecked()">
<input type="button" value="informa de su propiedad value" onclick="alertaValue()">
<br>
<br>
<input type="button" value="Simula un click" onclick="metodoClick()">
</form>
</body>
</html>
```

Para entender este acceso a las propiedades del checkbox tienes que apreciar que el formulario se llama "miFormulario". Este nombre se asigna en el atributo "name" de la etiqueta FORM.

Como el checkbox depende del formulario, podemos llegar a él mediante Javascript por medio de su nombre, nuevamente el atributo "name" pero esta vez del elemento "input checkbox". El name del elemento se indica con name="miCheck".

Otro medio de acceder a un elemento checkbox, mediante su id

Ahora vamos a ver un segundo ejemplo de manejo de elementos checkbox en Javascript. Es algo muy sencillo. Simplemente queremos mostrar el estado inicial de una casilla de verificación y, cuando se haga clic, mostrar el estado actual.

La gracia de este ejercicio es que te presenta una nueva manera de acceder a los elementos de formulario, que es más sencilla todavía que la anterior, porque solamente necesitamos saber el "id" del checkbox.

Este ejercicio se basa en que Javascript tiene un método de acceso directo a cualquier elemento del DOM (Document Object Model). Este método depende del objeto document y se llama document.getElementById(). Con este método podemos acceder a una casilla checkbox, pero también a cualquier otro elemento de la página, como un "div".

Veamos el código del ejemplo y lo comentamos a continuación.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>checkbox por id</title>
</head>
<body>

<input type="checkbox" id="miElementoCheckbox">

<div id="msg"></div>

<script>
  var miCheckbox = document.getElementById('miElementoCheckbox');
  var msg = document.getElementById('msg');

  alert('El valor inicial del checkbox es ' + miCheckbox.checked);
```

```
  miCheckbox.addEventListener('click', function() {
    if(miCheckbox.checked) {
      msg.innerText = 'El elemento está marcado';
    } else {
      msg.innerText = 'Ahora está desmarcado';
    }
  });
</script>

</body>
</html>
```

La parte interesante que comentábamos, del acceso al DOM la hacemos con las siguientes líneas:

```
var miCheckbox = document.getElementById('miElementoCheckbox');
var msg = document.getElementById('msg');
```

Hemos accedido al campo checkbox y a un div a través de sus atributos id. El elemento en sí, el objeto Javascript que lo representa, lo hemos guardado en su correspondiente variable.

Posteriormente puedo acceder a cualquier método o propiedad del elemento a través de la variable donde la hemos guardado.

```
miCheckbox.checked
```

Esa propiedad nos dirá si el elemento checkbox está o no marcado en ese instante.

Manejadodes de eventos en el checkbox con addEventListener

Quizás otra parte que merezca la pena comentar es cómo hemos asignado un manejador de evento al checkbox:

```
miCheckbox.addEventListener('click', function() {
  //
})
```

Una vez que tenemos un elemento del DOM cualquiera, podemos usar su método addEventListener para asociar manejadores de eventos. En este caso estamos definiendo un manejador para el evento "click", es decir, un código de una función que se ejecutará siempre que se haga clic.

Ahora, con este código que encuentras dentro del manejador de evento:

```
if(miCheckbox.checked) {
  msg.innerText = 'El elemento está marcado';
```

```
} else {
  msg.innerText = 'Ahora está desmarcado';
}
```

Simplemente estamos comprobando el estado del checkbox para cambiar el texto que hay dentro del elemento "msg" (el DIV que habíamos accedido a través de su id anteriormente), colocando un texto dependiendo de si está marcado o no el checkbox.

Eso es todo! esperamos que estos ejemplos de control de los checkboxes en Javascript te hayan servido para aprender a manejar estos campos de formulario.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *25/07/2021*
Disponible online en *https://desarrolloweb.com/articulos/995.php*

Controlar botones de radio con Javascript

Explicación y ejemplos sobre el manejo de radio buttons en Javascript. Exploraremos los métodos y propiedades de los botones de radio, junto con ejemplos de funcionamiento.

El botón de radio (o radio button en inglés) es un elemento de formulario que permite seleccionar una opción y sólo una, sobre un conjunto de posibilidades. Se puede ver a continuación.

Nota: En la parte de arriba podemos ver tres radio buttons en lugar de uno solo. Se colocan tres botones porque así podemos examinar su funcionamiento al formar parte de un grupo. Veamos que al seleccionar una opción se deselecciona la opción que estuviera marcada antes.

HTML para generar botones de radio

Aunque en este artículo nos interesa especialmente a trabajar con los botones de radio desde Javascript, queremos dejar claras un par de reglas del HTML para crear los botones de radio.

- Se consiguen con la etiqueta `<input type=radio>`.
- Con el atributo `name` de la etiqueta `<input>` les asignamos un nombre, que nos permite agrupar los radio button.
- Al tener el mismo `name` se consigue que sólo se pueda elegir una opción entre varias, dentro de ese grupo de botones de radio.
- Con el atributo `value` indicamos el valor de cada uno de los radio buttons.
- El atributo `checked` nos sirve para indicar cuál de los radio butons tiene que estar seleccionado por defecto.

Referencia: Explicamos en detalle la creación de botones de radio en nuestro manual de HTML, en el capítulo Otros elementos de formularios.

Objetos radio en Javascript

Ahora vamos dar una pequeña referencia de lo que podemos encontrar en Javascript para cada uno de los objetos de radio que crea el navegador en el DOM. Por medio de estas propiedades y métodos podemos manipular los objetos de radio dinámicamente.

De momento dejamos esta referencia aquí, pero más adelante te ponemos ejemplos concretos y prácticos sobre cómo manipular los radio buttons con Javascript.

Propiedades del objeto radio

Veamos una lista de las propiedades de este elemento.

checked

Indica si está chekeado o no un botón de radio.

defaultChecked

Su estado por defecto.

value

El valor del campo de radio, asignado por la propiedad value del radio.

Length

(como propiedad del array de radios)

El número de botones de radio que forman parte en el grupo. Accesible en el vector de radios.

Métodos del objeto radio

Son los mismos que los que tenía el objeto checkbox en Javascript.

Trabajo con botones de radio y Javascript

En Javascript accedemos a los elementos de la página mediante el DOM, que es una jerarquía de objetos que modelan cada uno de los elementos del HTML, es decir, las etiquetas que tiene una web.

Existen diversos modos de acceso a los elementos del DOM. De hecho, a medida que Javascript ha ido evolucionando, se han ido incorporando más métodos de acceso al DOM, por lo que existe una variedad de posibilidades de acceder a los botones de radio. En este artículo te vamos a explicar algunas de ellas, para que no tengas problemas por entender unos y otros métodos de trabajo con radio buttons en Javascript.

Método moderno de acceso al valor de los radio buttons con Javascript

Comenzamos por explicar el método más moderno, que es también el más directo de conseguir acceder a los botones de radio y saber cuál es el que está marcado.

Comenzamos viendo cómo acceder mediante el método querySelector que permite acceder a un objeto del DOM a través de un selector de los que usamos en CSS.

Vamos a partir de este HTML con una serie de botones de radio:

```
<input type="radio" name="status" value="interesado" id="interesado"> <label for="interesado">Estoy interesado</label>
<br>
<input type="radio" name="status" value="indeciso" id="indeciso"> <label for="indeciso">Estoy indeciso</label>
<br>
<input type="radio" name="status" value="no_interesado" id="no_interesado"> <label for="no_interesado">No me interesa</label>
```

Los botones de radio tienen todos el valor name="status", por tanto podremos acceder al valor

actual mediante este código:

```
valorActivo = document.querySelector('input[name="status"]:checked').value;
```

Sin embargo, esto tiene un problema y es que puede que un elemento no esté activo, en cuyo caso no se podría acceder al value de un null, puesto que sería un error en tiempo de ejecución.

Así pues, siendo un poco más cuidadosos podríamos tener un código como este:

```
let elementoActivo = document.querySelector('input[name="status"]:checked');
if(elementoActivo) {
    alert(elementoActivo.value);
} else {
    alert('No hay ninún elemento activo');
}
```

Método moderno para seleccionar un botón de radio con Javascript

Ahora vamos a ver cómo setear una opción determinada en un campo input radio. Para eso vamos a usar de nuevo un método moderno, basado en el método querySelectorAll().

El problema o la dificultad de marcar una opción determinada de los botones de radio es que tenemos varios elementos en la página, cada uno de los que forma el grupo. Por ello no es simplemente acceder a un objeto y marcarlo como checked, sino que tendremos que recorrer el conjuno de los botones de radio para encontrar el que nos interesa y entonces chequearlo.

Para facilitar este trabajo vamos a presentar una función que se encarga de realizar este proceso y podrás reutilizar allá donde lo necesites.

```
function setRadio(name, value) {
    document.querySelectorAll(`input[name="${name}"]`).forEach(element => {
        if(element.value === value) {
            element.checked = true;
        }
    });
}
```

Esta función se encarga de recibir dos parámetros:

- El nombre de los botones que deseas setear (el atributo name).
- El valor del input radio que se debe marcar como seleccionado

En el cuerpo de la función recorremos los botones de radio, seleccionando el que debería marcarse como activo. Para ello simplemente modificamos el valor de la propiedad checked del elemento que tiene el valor (value) que debe ser marcado.

Para que quede constancia, voy a dejar el código completo de una página que usa estos dos

mecanismos de acceso y manipulación de los botones de radio con Javascript.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Acceso a botones de radio con Javascript</title>
</head>
<body>
    <p>
        <input type="radio" name="status" value="interesado" id="interesado"> <label for="interesado">Estoy interesado</label>
        <br>
        <input type="radio" name="status" value="indeciso" id="indeciso"> <label for="indeciso">Estoy indeciso</label>
        <br>
        <input type="radio" name="status" value="no_interesado" id="no_interesado"> <label for="no_interesado">No me interesa</label>
    </p>
    <button id="mostrar">Mostrar opción activa</button>
    <button id="setear">Setear activo al valor "interesado"</button>

    <script>
        document.getElementById('mostrar').addEventListener('click', function() {
            // let valorActivo = document.querySelector('input[name="status"]:checked').value; // Esto tiene el problema de que puede que un elemento no esté activo, entonces no se podría acceder al
value de un null, lo que sería un error en tiempo de ejecución
            let elementoActivo = document.querySelector('input[name="status"]:checked');
            if(elementoActivo) {
                alert(elementoActivo.value);
            } else {
                alert('No hay ninún elemento activo');
            }
        });

        document.getElementById('setear').addEventListener('click', function() {
            setRadio('status', 'interesado')
        });

        function setRadio(name, value) {
            document.querySelectorAll(`input[name="${name}"]`).forEach(element => {
                if(element.value === value) {
                    element.checked = true;
                }
            });
        }
    </script>
</body>
</html>
```

Acceso a los botones de radio con el método tradicional basado en el objeto form

A continuación en este artículo te vamos a explicar un método más tradicional de acceso a los botones de radio, que era necesario antes de tener en Javascript los métodos querySelector() y querySelectorAll(). Funcionará todavía sin problema alguno, solamente tienes que tener en cuenta:

- Deberían estar los elementos de radio en un formulario.
- El formulario debe tener un nombre para poder referirnos a él.

Cuando en una página tenemos un conjunto de botones de radio se crea un objeto radio por cada uno de los botones. Los objetos radio dependen del formulario y se puede acceder a ellos por el array de elements del formulario (enseguida veremos cómo), sin embargo también se crea un array con los botones de radio. Este array depende del formulario y tiene el mismo nombre que los botones de radio.

Ejemplo de utilización de radio button con el método tradicional de Javascript basado en el objeto form

Veamos con un ejemplo el método de trabajo con los radio buttons en el que vamos a colocar un montón de ellos y cada uno tendrá asociado un color. También habrá un botón y al pulsarlo cambiaremos el color de fondo de la pantalla al color que esté seleccionado en el conjunto de botones de radio.

Vamos a ver la página entera y luego la comentamos.

```
<html>
<head>
	<title>Ejemplo Radio Button</title>
<script>
function cambiaColor() {
	var i
	for (i = 0; i < document.fcolores.colorin.length; i++){
	  if (document.fcolores.colorin[i].checked) {
	    break;
	  }
	}
	document.bgColor = document.fcolores.colorin[i].value
}
</script>
</head>

<body>
<form name=fcolores>
<input type="Radio" name="colorin" value="ffffff" checked> Blanco
<br>
<input type="Radio" name="colorin" value="ff0000"> Rojo
<br>
<input type="Radio" name="colorin" value="00ff00"> Verde
<br>
<input type="Radio" name="colorin" value="0000ff"> Azul
<br>
<input type="Radio" name="colorin" value="ffff00"> Amarillo
<br>
<input type="Radio" name="colorin" value="00ff00"> Turquesa
<br>
<input type="Radio" name="colorin" value="ff00ff"> Morado
<br>
<input type="Radio" name="colorin" value="000000"> Negro
<br>
<br>
<input type="Button" value="Cambia Color" onclick="cambiaColor()">
</form>
</body>
</html>
```

Primero podemos fijarnos en el formulario y en la lista de botones de radio. Todos se llaman "colorin", así que están asociados en un mismo grupo. Además vemos que el atributo value de cada botón cambia. También vemos un botón abajo del todo.

Con esta estructura de formulario tendremos un array de elements de 9 elementos, los 8 botones de radio y el botón de abajo.

Además tendremos un array de botones de radio que se llamará colorín y depende del formulario, accesible de esta manera.

```
document.form.colorin
```

Este array tiene en cada posición uno de los botones de radio. Así en la posición 0 está el botón del color blanco, en la posición 1 el del color rojo... Para acceder a esos botones de radio lo hacemos con su índice.

```
document.fcolores.colorin[0]
```

Si queremos acceder por ejemplo a la propiedad value del último botón de radio escribimos lo siguiente.

```
document.fcolores.colorin[7].value
```

La propiedad length del array de radios nos indica el número de botones de radio que forman parte del grupo.

```
document.fcolores.colorin.length
```

En este caso la propiedad length valdrá 8.

Con estas notas podremos entender más o menos bien la función que se encarga de encontrar el radio button seleccionado y cambiar el color de fondo de la página.

Algoritmo para saber qué botón de radio está seleccionado

Nuestro ejemplo de control de radiobuttons con Javascript es bastante sencillo. Todo comienza en un botón, que al hacer clic sobre él, desata la funcionalidad del cambio del color de la página.

```
<input type="Button" name="" value="Cambia Color" onclick="cambiaColor()">
```

Como puedes ver en el "onclick" del botón, invocamos a la función cambiaColor() que hace todo el trabajo.

Quizás la parte más compleja de entender, aunque sige siendo muy simple cuando ya tienes una ligera idea de programación y entiendes los elementos radio button, es el algoritmo que hemos utilizado para saber el índice del botón de radio que está seleccionado.

Dicho algoritmo sería esta parte del código:

```
var i
for (i = 0; i < document.fcolores.colorin.length; i++){
  if (document.fcolores.colorin[i].checked) {
     break;
 }
```

```
}
```

Explicamos paso a paso este sencillo código Javascript para averiguar qué botón de radio está pulsado:

1. Se declara una variable llamada "i" en la primera línea, que vamos a usar dentro del bucle.
2. Hacemos un bucle for que va desde i=0 hasta el número de elementos que hay en los radio button que queremos controlar. Este bucle permite ir recorriendo el array de botones de radio hasta que encontramos el que tiene su propiedad checked a true.
3. En cada iteración del bucle se guarda en la variable "i" el índice del radio button actual.
4. Dentro del bucle, se comprueba si el elemento de radio con ese índice tiene su propiedad checked a true. En ese momento salimos del bucle, con lo que la variable "i" almacenará el índice del botón de radio seleccionado.

Ya fuera del bucle, en la última línea de la función cambiaColor(), cambiamos el color de fondo. Se trata de este código:

```
document.bgColor = document.fcolores.colorin[i].value
```

Simplemente estamos cambiando la propiedad "bgColor" del objeto document al valor que hay en el atributo "value" del radio button seleccionado.

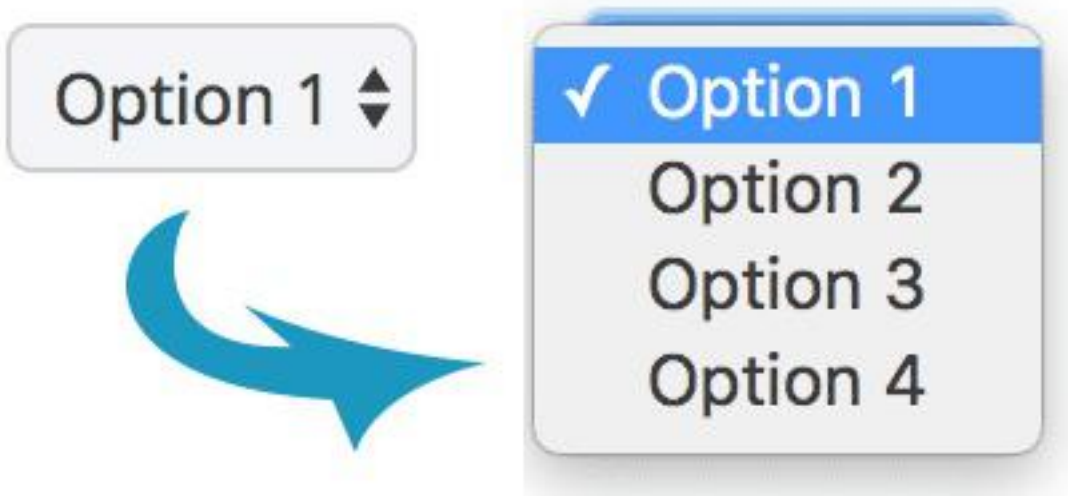
Referencia: Si deseamos profundizar el el control de botones de radio podemos acceder a un taller relacionado: Inhibir radio button con Javascript

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *15/06/2022*
Disponible online en *https://desarrolloweb.com/articulos/1006.php*

Control de campos select con Javascript

Descripción y ejemplo para el control de campos de formulario select, también llamados listas desplegables o combo box.

El objeto select de un formulario es una de esas listas desplegables que nos permiten seleccionar un elemento. Se despliegan apretando sobre una flecha, a continuación se puede escoger un elemento y para acabar se vuelven a plegar. Se puede ver un elemento select de un formulario a continuación.



Uno de estos elementos se puede obtener utilizando la etiqueta <SELECT> dentro de un formulario. A la etiqueta le podemos añadir un atributo para darle el nombre, NAME, para luego acceder a ese campo mediante Javascript. Para expresar cada una de las posibles opciones del campo select utilizamos una etiqueta <OPTION> a la que le introducimos un atributo VALUE para expresar su valor. El texto que colocamos después de la etiqueta <OPTION> sirve como etiqueta de esa opción, es el texto que ve el usuario asociado a esa opción.

Propiedades del objeto select

Vamos a ver una lista de las propiedades de este elemento de formulario.

length

Guarda la cantidad de opciones del campo select. Cantidad de etiquetas <OPTION>

Option

Hace referencia a cada una de sus opciones. Son por si mismas objetos.

options

Un array con cada una de las opciones del select.

selectedIndex

Es el índice de la opción que se encuentra seleccionada.

Aparte de las conocidas propiedades comunes a todos los elementos de formulario: form y name y type.

Métodos del objeto select

Los métodos son solamente 2 y ya conocemos su significado.

blur()

Para retirar el foco de la aplicación de ese elemento de formulario.

focus()

Para poner el foco de la aplicación.

Objeto option

Tenemos que pararnos a ver también este objeto para entender bien el campo select. Recordamos que las option son las distintas opciones que tiene un select, expresadas con la etiqueta <OPTION>.

Propiedades de option

Estos objetos sólo tienen propiedades, no tienen métodos. Vamos a verlas.

defaultSelected

Indica con un true o un false si esa opción es la opción por defecto. La opción por defecto se consigue con HTML colocando el atributo selected a un option.

index

El índice de esa opción dentro del select.

selected

Indica si esa opción se encuentra seleccionada o no.

text

Es el texto de la opción. Lo que puede ver el usuario en el select, que se escribe después de la etiqueta <OPTION>.

value

Indica el valor de la opción, que se introduce con el atributo VALUE de la etiqueta <OPTION>.

Ejemplo de acceso a un select

Vamos a ver un ejemplo sobre cómo se accede a un select con Javascript, como podemos

acceder a sus distintas propiedades y a la opción seleccionada.

Vamos a empezar viendo el formulario que tiene el select con el que vamos a trabajar. Es un select que sirve para valorar el web que estamos visitando.

```
<form name="fomul">
Valoración sobre este web:
<select name="miSelect">
<option value="10">Muy bien
<option value="5" selected>Regular
<option value="0">Muy mal
</select>
<br>
<br>
<input type=button value="Dime propiedades" onclick="dimePropiedades()">
</form>
```

Ahora vamos a ver una función que recoge las propiedades más significativas del campo select y las presenta en una caja alert.

```
function dimePropiedades(){
    var texto
    texto = "El numero de opciones del select: " + document.formul.miSelect.length
    var indice = document.formul.miSelect.selectedIndex
    texto += "nIndice de la opcion escogida: " + indice
    var valor = document.formul.miSelect.options[indice].value
    texto += "nValor de la opcion escogida: " + valor
    var textoEscogido = document.formul.miSelect.options[indice].text
    texto += "nTexto de la opcion escogida: " + textoEscogido
    alert(texto) }
```

Esta función crea una variable de texto donde va introduciendo cada una de las propiedades del select. La primera contiene el valor de la propiedad length del select, la segunda el índice de la opción seleccionada y las dos siguientes contienen el valor y el texto de la opción seleccionada. Para acceder a la opción seleccionada utilizamos el array options con el índice recogido en las segunda variable.

Podemos ver el ejemplo en funcionamiento aquí.

Podemos ver un ejemplo más practico sobre qué se puede hacer con un campo select en el reportaje de cómo hacer un navegador desplegable con Javascript.

Propiedad value de un objeto select

Para acceder al valor seleccionado de un campo select podemos utilizar la propiedad value del campo select en lugar de acceder a partir del vector de options.

Para el anterior formulario sería algo parecido a esto.

```
formul.miSelect.value
```

Sin embargo, esta propiedad sólo está presente en navegadores Internet Explorer, por lo que es

recomendable acceder utilizando el vector de options con el índice de la posición seleccionada si deseamos que la página sea compatible con todos los navegadores. Hemos querido añadir este punto para que, si alguna vez utilizamos o vemos utilizar este método de acceso, sepamos su pega y porque es mejor utilizar el vector de options.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *15/01/2003*
Disponible online en *https://desarrolloweb.com/articulos/1027.php*

Control de elementos Textarea en Javascript

Los elementos textarea son los campos que permiten introducir varias líneas de texto. Aprendemos a controlarlos con programación Javascript.

Para acabar de describir todos los elementos de formularios vamos a ver el objeto textarea que es un elemento que presenta un lugar para escribir texto, igual que los campos text, pero con la particularidad que podemos escribir varias líneas a la vez.

El campo textarea se puede ver a continuación.

Un campo textarea se consigue con la etiqueta <TEXTAREA>. Con el atributo name le podemos dar un nombre para acceder al campo textarea mediante Javascript. Otros atributos interesantes son cols y rows que sirven para indicar la anchura y altura del campo textarea en caracteres, cols indica el número de columnas y rows el de filas. aunque no se puede acceder a ellos con Javascript. El valor por defecto de un campo textarea se coloca entre las etiquteta <TEXTAREA> y su correspondiente cierre.

Propiedades de textarea

Se puede ver una lista de las propiedades disponibles en un textarea a continuación, que son los mismos que un campo de texto.

defaultValue

Que contiene el valor por defecto del textarea.

value

Que contiene el texto que hay escrito en el textarea.

Además tiene las conocidas propiedades de elementos de formulario form, name y type.

Métodos de textarea

Veamos una lista de los métodos, que son los mismos que en un campo de texto.

blur()

Para quitar el foco de la aplicación del textarea.

focus()

Para poner el foco de la aplicación en el textarea.

select()

Selecciona el texto del textarea.

Vamos a ver un ejemplo a continuación que presenta un formulario que tiene un textarea y un botón. Al apretar el botón se cuenta el número de caracteres y se coloca en un campo de texto.

Para acceder al número de caracteres lo hacemos a partir del la propiedad value del objeto textarea. Como value contiene un string podemos acceder a la propiedad length que tienen todos los strings para averiguar su longitud.

El código de la página se puede ver aquí.

```
<html>
<head>
      <title>Ejemplo textarea</title>
<script>
function cuenta(){
   numCaracteres = document.formul.textito.value.length
      document.formul.numCaracteres.value = numCaracteres
}
</script>
</head>
<body>
<form name="formul">
<textarea name=textito cols=40 rows=3>
Este es el texto por defecto
</textarea>
<br>
<br>
Número de caracteres <input type="Text" name="numCaracteres" size="4">
<br>
<br>
<input type=button value="Cuenta caracteres" onclick="cuenta()">
</form>
</body>
</html>
```

El ejemplo funcionando se puede ver en una página independiente.

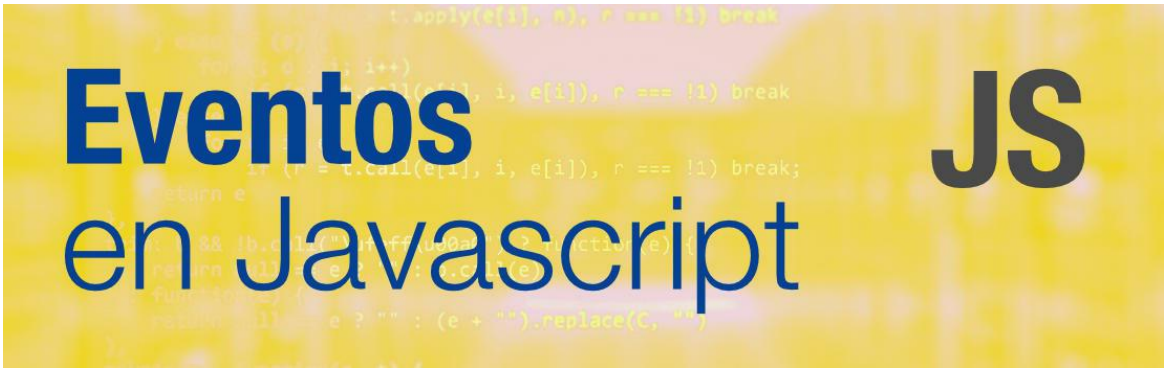
Para quien desee profundizar, tenemos un artículo interesante que amplía el ejemplo anterior. Se trata de una cuenta de los caracteres del textarea a la vez que se está escribiendo dentro del campo. Se realiza gracias al tratamiento de eventos. Se puede ver el artículo en la dirección http://www.desarrolloweb.com/articulos/1348.php

Eventos en Javascript

Los eventos son la base de la interactividad, de modo que en este apartado veremos cómo hacer páginas que respondan a las acciones del usuario. En Javascript podemos ejecutar código como respuesta a eventos, que son distintos tipos de acciones que el visitante puede realizar sobre la página o sobre sus elementos.

Los eventos en Javascript

Una explicación sobre lo que son los eventos en Javascript, con conceptos básicos que debemos conocer y ejemplos de eventos y manejadores de eventos en Javascript.



Los eventos son la manera que tenemos en Javascript de controlar las acciones de los visitantes y definir un comportamiento de la página cuando se produzcan. Cuando un usuario visita una página web e interactúa con ella se producen los eventos y con Javascript podemos definir qué queremos que ocurra cuando se produzcan los eventos.

Para entender los eventos necesitamos conocer algunos conceptos básicos:

- Evento: Es algo que ocurre. Generalmente los eventos ocurren cuando el usuario interactúa con el documento, pero podrían producirse por situaciones ajenas al usuario, como una imagen que no se puede cargar porque esté indisponible.
- Tipo de evento: Es el tipo del suceso que ha ocurrido, por ejemplo, un clic sobre un botón o el envío de un formulario. Cada tipo de elemento de la página ofrece diversos tipos de eventos de Javascript. En esta página puedes saber cuáles son los tipos de eventos en Javascript.
- Manejador de evento: es el comportamiento que nosotros pondemos asignar como respuesta a un evento. Se especifica mediante una función Javascript, que se asocia a un tipo de evento en concreto. Una vez asociado el manejador a un tipo de evento sobre un elemento de la página, da vez que ocurre ese tipo de evento sobre ese elemento en concreto, se ejecutará el manejador de evento asociado.

Por qué necesitamos los eventos en Javascript

Con javascript podemos definir qué es lo que pasa cuando se produce un evento, como podría ser que un usuario pulse sobre un botón, edite un campo de texto o abandone la página.

El manejo de eventos es el caballo de batalla para hacer páginas interactivas, porque con ellos podemos responder a las acciones de los usuarios. Hasta ahora en este manual hemos podido ver muchos ejemplos de manejo de uno de los eventos de Javascript, el evento "click", que se produce al pulsar un elemento de la página. Hasta ahora siempre hemos aplicado el evento a un botón, pero podríamos aplicarlo a otros elementos de la página.

Cómo se define un evento en Javascript

Para definir las acciones que queremos realizar al producirse un evento utilizamos los manejadores de eventos. Existen muchos tipos de eventos sobre los que podemos asociar manejadores de eventos, para muchos tipos de acciones del usuario.

En Javascript podemos definir eventos de dos maneras distintas. Una manera es en el propio código HTML, usando atributos de los elementos (etiquetas) a los que queremos asociar los manejadores de eventos. Otra manera un poco más avanzada es usando los propios objetos del DOM. Vamos a ver ambas maneras a continuación.

Manejadores de evento especificados en el código HTML

El maneador de eventos se puede colocar en la etiqueta HTML del elemento de la página que queremos que responda a las acciones del usuario. Para ello usamos atributos especiales en las etiquetas del HTML, que tienen el prefijo "on" seguido del tipo de evento. Por ejemplo el manejador asociado en el atributo "onclick" se ejecuta cuando se produce un clic en una etiqueta.

Vamos a poner un ejemplo sobre el manejador de eventos onclick. Ya sabemos que sirve para describir acciones que queremos que se ejecuten cuando se hace un clic. Por tanto, si queremos que al hacer click sobre un botón pase alguna cosa, escribimos el manejador onclick en la etiqueta <INPUT type=button> de ese botón. Algo parecido a esto.

```
<INPUT type="button" value="pulsame" onclick="sentencias_javascript...">
```

Se coloca un atributo nuevo en la etiqueta que tiene el mismo nombre que el evento, en este caso onclick. El atributo se iguala a las sentencias Javascript que queremos que se ejecuten al producirse el evento.

```
<INPUT type="button" value="pulsame" onclick="alert('has hecho clic')">
```

Dado el código anterior, al hacer clic sobre el botón aparecerá un mensaje de alerta con el texto "has hecho clic".

Cada elemento de la página tiene su propia lista de eventos soportados, vamos a ver otro ejemplo de manejo de eventos, esta vez sobre un menú desplegable, en el que definimos un comportamiento cuando cambiamos el valor seleccionado.

```
<SELECT onchange="window.alert('Cambiaste la selección')">
<OPTION value="opcion1">Opcion 1
<OPTION value="opcion2">Opcion 2
</SELECT>
```

> window.alert('mensaje') equivale a alert('mensaje'), ya que todos los métodos que pertenecen al objeto Window se pueden ejecutar sin referenciar al objeto window. Esto ya lo aprendimos al hablar de la jerarquía de objetos del navegador.

Dentro de los manejadores de eventos podemos colocar tantas instrucciones como deseemos, pero siempre separadas por punto y coma. Lo habitual es colocar una sola instrucción, y si se desean colocar más de una se suele crear una función con todas las instrucciones y dentro del manejador se coloca una sola instrucción que es la llamada a la función.

Vamos a ver cómo se colocarían en un manejador varias instrucciones.

```
<input type=button value=Pulsame
  onclick="x=30; window.alert(x); window.document.bgColor = 'red'">
```

Son instrucciones muy simples como asignar a x el valor 30, hacer una ventana de alerta con el valor de x y cambiar el color del fondo a rojo. Podemos ver el ejemplo en una página aparte.

Sin embargo, tantas instrucciones puestas en un manejador quedan un poco confusas, habría sido mejor crear una función así.

```
<script>
function ejecutaEventoOnclick(){
      var x = 30;
      window.alert(x);
      window.document.bgColor = 'red';
}
</script>

<FORM>
<input type="button" value="Pulsame" onclick="ejecutaEventoOnclick()">
</FORM>
```

Ahora utilizamos más texto para hacer lo mismo, pero seguro que a la mayoría les parece más claro este segundo ejemplo. Siempre es una idea hacer un código mantenible y poner varias instrucciones en un atributo onclick no es una buena idea.

######### Jerarquía desde el objeto window

En los manejadores de eventos se tiene que especificar la jerarquía entera de objetos del

navegador, empezando siempre por el objeto window. Esto es necesario porque hay algún browser antiguo que no sobreentiende el objeto window cuando se escriben sentencias Javascript vinculadas a manejadores de eventos.

Manejadores de eventos asociados con addEventListener

La segunda forma de asociar manejadores de eventos a elementos de la página es mediante el método addEventListener(). Es una forma un poco más avanzada, pero mejora todavía la mantenibilidad del código, ya que permite separar mejor el código de la funcionalidad del código del contenido.

> El HTML debería usarse simplemente para definir el contenido. Si tenemos instrucciones Javascript dentro de las etiquetas, colocando atributos como "onclick" o "onchange" lo que estamos haciendo es colocar código de la funcionalidad dentro del código HTML, lo que es poco deseable desde el punto de vista de separación de responsabilidades. Por tanto, la técnica que vamos a conocer ahora es todavía más adecuada, porque nos va a permitir escribir el código de la funcionalidad (los eventos javascript) sin ensuciar el código HTML.

Para asociar un evento a un elemento de la página necesitamos hacer dos pasos:

1. Acceder al objeto sobre el que queremos definir el evento. Para esto tenemos que acceder al DOM para localizar el objeto adecuado, que representa la etiqueta sobre la que queremos asociar el manejador del evento.
2. Sobre el objeto del DOM, aplicamos addEventListener(), indicando el tipo de evento y la función manejadora.

Por ejemplo tenemos este elemento de la página:

```
<input type="button" id="elboton" value="Haz click">
```

Lo más cómodo para acceder a un elemento de la página y recuperar el objeto del DOM asociado a esa etiqueta es usar el identificador (atributo "id"). En este caso el identificador es "elboton". Para acceder a ese elemento usamos el método getElementById() del objeto document, enviando el identificador.

```
var miBoton = document.getElementById('elboton');
```

Ahora tenemos el objeto del DOM asociado a ese botón en la variable "miBoton". Sobre ese objeto ya podemos invocar el método addEventListener(). A este método debemos pasarle dos parámetros. El primero es el tipo de evento que queremos detectar y el segundo la función manejadora del evento que queremos que se ejecute cuando se produce el evento.

```
miBoton.addEventListener('click', function() {
```

```
  alert('Has hecho clic!!')
})
```

Ya lo tenemos! al hacer clic sobre el botón se mostrará un mensaje de alerta.

Vamos a ver un segundo ejemplo, sobre una imagen en la que vamos a asociar un manejador para el tipo de evento "mouseover", que se produce cuando el usuario pone el puntero del ratón encima de un elemento.

Tenemos una imagen, en la que hemos puesto un atributo id para llegar a ella.

```
<img src="https://picsum.photos/200/300" id="imagen">
```

Ahora le asociamos el manejador de evento para el tipo de evento "mouseover" con este código Javascript.

```
var miImagen = document.getElementById('imagen');
miImagen.addEventListener('mouseover', function() {
  alert('Has pasado el ratón encima de la imagen')
})
```

Objeto evento

Solo queremos dar un último detalle sobre eventos abordando el objeto evento, que recibimos en todas las funciones que hacen de manejadores de eventos. Mediante el objeto evento podemos recibir multitud de datos útiles sobre las condiciones en las cuales un evento se ha producido.

El objeto evento de Javascript se genera automáticamente por parte del navegador y nos lo envían como parámetro a la función asociada al evento. En muchas ocasiones, como en los ejemplos anteriores, no necesitamos el objeto evento para saber nada en concreto, por lo que simplemente podemos omitir ese parámetro y no recibir ese objeto evento, pero si lo necesitamos para algo, simplemente colocamos el parámetro al definir el manejador.

```
miImagen.addEventListener('mouseover', function(objetoEvento) {
  console.log(objetoEvento)
})
```

En el código anterior simplemente hemos modificado la función manejadora para recibir el objeto evento. Dentro de la función manejadora simplemente estamos enviando ese objeto a la consola, con lo que podemos ver la cantidad de datos que nos ofrece.

```
                                                                    VM1086:2
MouseEvent {isTrusted: true, screenX: 714, screenY: 390, clientX: 787, clientY: 317, …}
  isTrusted: true
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 787
  clientY: 317
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 0
  eventPhase: 0
  fromElement: null
  layerX: 787
  layerY: 7091
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 788
  offsetY: 7091
  pageX: 787
  pageY: 7091
▶ path: (4) [body, html, document, Window]
  relatedTarget: null
  returnValue: true
  screenX: 714
  screenY: 390
  shiftKey: false
▶ sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
▶ srcElement: body
▶ target: body
  timeStamp: 1009309
▶ toElement: body
  type: "mouseover"
▶ view: Window {0: global, 1: global, 2: global, 3: global, 4: global, 5: global, 6: Win…
  which: 0
  x: 787
  y: 317
▶ [[Prototype]]: MouseEvent
```

Ahora vamos a ver un código de definición de un evento sobre el objeto document, en el cual vamos a ver dónde han hecho clic dentro de un documento, las coordenadas exactas. Para ello usamos las propiedades clientX y clientY del objeto evento, tal como puedes ver en este código.

```
document.addEventListener('click', function(event) {
  alert('Has hecho clic en ' + event.clientX + 'x' + event.clientY);
})
```

Un detalle interesante es que este evento lo hemos asociado al objeto document directamente, por lo que el evento clic se está detectando en todo el documento HTML, así que pulses donde pulses se mostrará la caja de alerta informando de las coordenadas.

Como el objeto document está siempre disponible en el navegador, solamente hemos tenido que acceder a document.addEventListener(), mientras que antes necesitábamos hacer un paso previo para traernos el elemento al que queríamos asociar el manejador con getElementById().

Conclusión

Hemos aprendido bastantes cosas sobre los eventos en Javascript, sobre todo te debe quedar claro qué es un evento y qué es un manejador de evento. Además hemos visto cómo podemos asociar manejadores de eventos a cualquier elemento de la página, de dos modos, a través de atributos en las etiquetas y a través de addEventListener(), siendo más avanzaza y conveniente esta segunda técnica.

En el siguiente artículo te vamos a explicar cuáles son los tipos de eventos más importantes en Javascript, de modo que tendrás una idea más global sobre las cosas que vas a poder detectar en la página y asociar los correspondientes manejadores de eventos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *28/12/2021*
Disponible online en *https://desarrolloweb.com/articulos/1235.php*

Los tipos de eventos en Javascript

Lista de los tipos de eventos más habituales en el lenguaje Javascript, junto con una descripción de cada uno.

Ahora vamos a ver una lista de los tipos de eventos que hay disponibles en Javascript, ofreciendo una pequeña descripción de cada uno.

Los tipos de eventos los define el estándar Javascript y se implementan en los navegadores. Contienen todas las situaciones o cosas destacables que el propio navegador te va a informar. Nosotros podremos crear manejadores de eventos a todos estos tipos de eventos. Lógicamente, dependiendo de la aplicación web que estés realizando y lo que quieras conseguir, crearás manejadores a unos tipos de eventos u otros, bajo tus propios criterios o necesidades.

Si queremos saber previamente qué es un evento y como se tratan en Javascript, podemos consultar el artículo anterior del manual: Los eventos en Javascript

Tipos de eventos en Javascript 1.2

Estos tipos de eventos son los más comunes, presentes en Javascript 1.2. Existen otros tipos de eventos que también son muy interesantes y veremos a continuación en este mismo artículo.

Cada evento tiene un nombre, por ejemplo "click". Además en el propio código HTML podemos asociar manejadores de eventos mediante atributos, que son usados para invocar una

serie de comandos cuando se produce un evento sobre ese elemento. Los atributos que podemos usar en el HTML para asociar manejadores de eventos tienen siempre el prefijo "on" seguido del nombre del evento. Por ejemplo, "onclick".

> Nota: La lista de tipos de eventos contiene el nombre del manejador en negrita, su descripción y finalmente la versión de Javascript que incorporó dicho manejador.

abort (onabort)

Este evento se produce cuando un usuario detiene la carga de una imagen, ya sea porque detiene la carga de la página o porque realiza una acción que la detiene, como por ejemplo irse de la página. Javascript 1.1

blur (onblur)

Se desata un evento onblur cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento. Javascript 1.0

change (onchange)

Se desata este evento cuando cambia el estado de un elemento de formulario, en ocasiones no se produce hasta que el usuario retira el foco de la aplicación del elemento. Javascript 1.0 Javascript 1.0

click (onclick)

Se produce cuando se da una pulsación o clic al botón del ratón sobre un elemento de la página, generalmente un botón o un enlace. Javascript 1.0

dragdrop (ondragdrop)

Se produce cuando un usuario suelta algo que había arrastrado sobre la página web. Javascript 1.2

error (onerror)

Se produce cuando no se puede cargar un documento o una imagen y esta queda rota. Javascript 1.1

focus (onfocus)

El evento onfocus es lo contrario de onblur. Se produce cuando un elemento de la página o la

ventana ganan el foco de la aplicación. Javascript 1.0

keydown (onkeydown)

Este evento se produce en el instante que un usuario presiona una tecla, independientemente que la suelte o no. Se produce en el momento de la pulsación. Javascript 1.2

keypress (onkeypress)

Ocurre un evento onkeypress cuando el usuario deja pulsada una tecla un tiempo determinado. Antes de este evento se produce un onkeydown en el momento que se pulsa la tecla.. Javascript 1.2

keyup (onkeyup)

Se produce cuando el usuario deja de apretar una tecla. Se produce en el momento que se libera la tecla. Javascript 1.2

load (onload)

Este evento se desata cuando la página, o en Javascript 1.1 las imágenes, ha terminado de cargarse. Javascript 1.0

mousedown (onmousedown)

Se produce el evento onmousedown cuando el uuario pulsa sobre un elemento de la página. onmousedown se produce en el momento de pulsar el botón, se suelte o no. Javascript 1.2

mousemove (onmousemove)

Se produce cuando el ratón se mueve por la página. Javascript 1.2

mouseout (onmouseout)

Se desata un evento onmuoseout cuando el puntero del ratón sale del área ocupada por un elemento de la página. Javascript 1.1

mouseover (onmouseover)

Este evento se desata cuando el puntero del ratón entra en el área ocupada por un eolemento de la página. Javascript 1.0

mouseup (onmouseup)

Este evento se produce en el momento que el usuario suelta el botón del ratón, que

previamente había pulsado. Javascript 1.2

move (onmove)

Evento que se ejecuta cuando se mueve la ventana del navegador, o un frame. Javascript 1.2

resize (onresize)

Evento que se produce cuando se redimensiona la ventana del navegador, o el frame, en caso de que la página los tenga. Javascript 1.2

reset (onreset)

Este evento está asociado a los formularios y se desata en el momento que un usuario hace clic en el botón de reset de un formulario. Javascript 1.1

select (onselect)

Se ejecuta cuando un usuario realiza una selección de un elemento de un formulario. Javascript 1.0

submit (onsubmit)

Ocurre cuando el visitante apreta sobre el botón de enviar el formulario. Se ejecuta antes del envío propiamente dicho. Javascript 1.0

unload (onunload)

Al abandonar una página, ya sea porque se pulse sobre un enlace que nos lleve a otra página o porque se cierre la ventana del navegador, se ejecuta el evento onunload. Javascript 1.0

Tipos de eventos en Javascript 1.3

Hasta ahora en el artículo vimos los eventos de las versiones de Javascript 1.0, 1.1 y 1.2, pero en este caso vamos a presentar el listado de los manejadores de eventos de Javascript 1.3.

DblClick (ondblclick):

Este evento es lanzado cuando el usuario hace doble click en un elemento de formulario o un link.

scroll (onscroll):

Este evento se produce cuando se realiza scroll o desplazamiento. Este desplazamiento se puede realizar en la página completa, pero también en elementos de la página que tengan sus

propias zonas de scroll.

Conclusión

Hemos conocido los tipos de eventos más usados en Javascript. Es solo un listado de los más importantes. Existen otros que no son tan utilizados pero que podrías consultar en la documentación del DOM del elemento en concreto que necesites controlar.

No te preocupes si algún conocimiento ha quedado en el aire, porque vamos a seguir haciendo prácticas con eventos en Javascript. Por ejemplo, en el siguiente artículo verás un ejemplo del evento abort. El manual también está plagado de ejemplos de eventos a continuación.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *28/12/2021*
Disponible online en *https://desarrolloweb.com/articulos/1236.php*

Ejemplos de eventos en Javascript. Onabort

Vemos enlaces a diversas aplicaciones prácticas donde se tratan eventos y ofrecemos un nuevo ejemplo para el evento onabort.

A lo largo de los manuales I y II de Javascript, así como del Taller, hemos mostrado muchos ejemplos de utilización de los manejadotes de eventos. Aquí veremos ejemplos sencillos que se nos ocurren para utilizar otros manejadotes que no hemos visto todavía, aunque antes podemos hacer una lista de algunos ejemplos publicados anteriormente que deberían servir de ayuda para ir captando la práctica de el manejo de eventos.

- Acceso por clave con Javascript (Evento onclick)
- Rollover con Javascript (Eventos onmuoseover y onmouseout)
- Navegador desplegable (Evento onchange)
- Calculadora sencilla (Evento onclick)
- Confirmación del envío de formulario (Evento onclick)
- Posicionarse en un select (Evento onkeypress)
- Inhibir campo de formulario (Evento onfocus)
- Cuenta caracteres de un textarea (Eventos onkeydown y onkeyup)

Evento onabort

Veamos un primer ejemplo, en este caso sobre el evento onabort. Este evento se activa al cancelarse la carga de una página, ya sea porque se pulsa el botón de cancelar o porque el usuario se marcha de la página por otro enlace.

Este ejemplo contiene una imagen que tiene el evento onabort asignado para que se ejecute una función en caso de que la imagen no llegue a cargarse. La función informa al usuario de que la imagen no se ha llegado a cargar y le pregunta si desea cargarla otra vez. Si el usuario contesta que sí, entonces se pone a descargar la imagen otra vez. Si dice que no, no hace nada.

La pregunta se hace con una caja confirm de Javascript.

```
<html> <head>
    <title>Evento onabort</title>

<script>
function preguntarSeguir(){
    respuesta = confirm ("Has detenido la carga de la página y hay una imagen que no estás viendo.n¿Deseas cargar la imagen?")
    if (respuesta)
      document.img1.src = "http://ipaginate.iespana.es/ipaginate/desarrollogrande.gif"
}
</script>

</head>
<body>
<img name=img1 src="http://ipaginate.iespana.es/ipaginate/desarrollogrande.gif" width=500 height=458 alt="Imagen que pesa 115K" border="0" onabort="preguntarSeguir()">
<br>
Pulsa el botón de parar la carga de la página y se pondrá en marcha el evento onerror

</body>
</html>
```

Este ejemplo estaría bien si siempre se detuviese la carga por pulsar el botón de cancelar, pero si lo que pasa es que el usuario ha cancelado por irse a otra página a través de un enlace, saldrá la caja de confirmación pero no ocurrirá nada independientemente de lo que se responda y el navegante se marchará irremediablemente a la nueva página.

Se puede ver en una página aparte.

Ejemplo del evento onblur en Javascript

Script en Javascript que muestra el trabajo con el evento onblur. Se comprueba la validez de un dato al salir del campo de texto donde está escrito.

Onblur se activa cuando el usuario retira el foco de la aplicación de un elemento de la página. El foco de la aplicación es el lugar donde está el cursor.

Si por ejemplo, estamos situados en un campo de texto y nos vamos de dicho campo, ya sea porque pulsamos con el ratón en otro campo del formulario o en un área vacía, ya sea porque el usuario a apretado el botón tabulador (Tab) que mueve el foco hasta el siguiente elemento de la página.

Si yo deseo que, al situarse fuera de un campo de texto, se compruebe que el valor introducido es correcto puedo utilizar onblur y llamar a una función que compruebe si el dato es correcto. Si no es correcto puedo obligar al foco de la aplicación a situarse nuevamente sobre el campo de texto y avisar al usuario para que cambie dicho valor.

Puede ser una manera interesante de asegurarnos que en un campo de texto se encuentra un valor válido. Aunque tiene alguna pega, como veremos más adelante.

Vamos a empezar por un caso sencillo, en el que solamente deseamos comprobar un campo de texto para asegurarnos que es un número entero.

> Referencia: La función que valida un entero la hemos explicado en un taller anterior de Javascript: Validar entero en campo de formulario.

```
<html>
<head>
	<title>Evento onblur</title>

<script>
function validarEntero(valor){
   //intento convertir a entero.
   //si era un entero no le afecta, si no lo era lo intenta convertir
   valor = parseInt(valor)

   //Compruebo si es un valor numérico
   if (isNaN(valor)) {
      //entonces (no es numero) devuelvo el valor cadena vacia
      return ""
   }else{
      //En caso contrario (Si era un número) devuelvo el valor
      return valor
   }
}

function compruebaValidoEntero(){
	enteroValidado = validarEntero(document.f1.numero.value)
	if (enteroValidado == ""){
	   //si era la cadena vacía es que no era válido. Lo aviso
	   alert ("Debe escribir un entero!")
	   //selecciono el texto
	   document.f1.numero.select()
	   //coloco otra vez el foco
	   document.f1.numero.focus()
	}else
	   document.f1.numero.value = enteroValidado
}
</script>
</head>
<body>
<form name=f1>
Escriba un número entero: <input type=text name=numero size=8 value="" onblur="compruebaValidoEntero()">
</form>

</body>
</html>
```

Al salirse del campo de texto (onblur) se ejecuta compruebaValidoEntero(), que utiliza la función validarEntero, explicada en un taller de Javascript. Si el valor devuelto por la función no es el de un entero, en este caso se recibiría una cadena vacía, muestra un mensaje en una caja alert, selecciona el texto escrito en la caja y coloca el foco de la aplicación en la caja de texto, para que el usuario coloque otro valor.

Hasta que el visitante no escriba un número entero en el campo de texto el foco de la

aplicación permanecerá en el campo y continuará recibiendo mensajes de error.

Podemos ver este ejemplo en marcha en una página aparte.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *25/08/2003*
Disponible online en *https://desarrolloweb.com/articulos/1251.php*

Continuación del ejemplo de onblur, para validar varios campos de texto

Hacemos un ejemplo de validación de campos de un formulario utilizando como base el evento onblur y solucionando un problema de bucle infinito.

Hemos visto en el ejemplo del método onblur relatado anteriormente una posible técnica para comprobar los datos de un campo de formulario. Ahora vamos a ver cómo evolucionar esta técnica si tenemos más de un campo a validar, dado que se puede complicar bastante el problema.

De hecho, antes de leer nuestra solución propuesta, creo que sería un buen ejercicio a realizar por el lector la práctica de hacer ese mismo ejemplo para dos campos y trabajar un poco con la página a ver si encontramos algún problema.

Muy probablemente nos encontraremos con un curioso bucle infinito que nos va a dar más de un quebradero de cabeza para solucionarlo.

En la práctica, el lector puede intentar validar un número entero y un código postal. Para validar un código postal necesitamos comprobar que es una cadena de texto compuesta por 5 caracteres y todos son enteros, por lo menos para los códigos en España.

Por si alguien lo quiere intentar, la función para validar un código postal sería algo parecido a esto:

```
function ValidoCP(){
      CPValido=true
      //si no tiene 5 caracteres no es válido
  if (document.f1.codigo.value.length != 5)
        CPValido=false
    else{
      for (i=0;i<5;i++){
        CActual = document.f1.codigo.value.charAt(i)
        if (validarEntero(CActual)==""){
          CPValido=false
          break;
        }
      }
    }
  return CPValido
}
```

Simplemente se encarga de comprobar que el campo de texto contiene 5 caracteres y hacer un

recorrido por cada uno de los caracteres para comprobar que todos son enteros. Al principio se supone que el código postal es correcto, colocando la variable CPValido a true, y si alguna comprobación falla se cambia el estado correcto a incorrecto, pasando dicha variable a false.

Se puede probar a montar el ejemplo con dos campos... nosotros ahora vamos a dar una solución al problema bastante complicadilla, ya que incluimos instrucciones para evitar el efecto del bucle infinito. No vamos a ver el ejemplo que daría el error, lo dejamos para el que desee intentarlo por si mismo y recomendamos hacerlo porque así comprenderemos mejor el siguiente código.

```
<html>
<head>
      <title>Evento onblur</title>

<script>
avisado=false
function validarEntero(valor){
     //intento convertir a entero.
     //si era un entero no le afecta, si no lo era lo intenta convertir
      valor = parseInt(valor)

     //Compruebo si es un valor numérico
     if (isNaN(valor)) {
       //entonces (no es numero) devuelvo el valor cadena vacia
       return ""
     }else{
       //En caso contrario (Si era un número) devuelvo el valor
       return valor
     }
}

function compruebaValidoEntero(){
  enteroValidado = validarEntero(document.f1.numero.value)
  if (enteroValidado == ""){
    //si era la cadena vacía es que no era válido. Lo aviso
    if (!avisado){
      alert ("Debe escribir un entero!")
      //selecciono el texto
      document.f1.numero.select()
      //coloco otra vez el foco
      document.f1.numero.focus()
      avisado=true
      setTimeout('avisado=false',50)
    }
  }else
    document.f1.numero.value = enteroValidado
}

function compruebaValidoCP(){
  CPValido=true
  //si no tiene 5 caracteres no es válido
  if (document.f1.codigo.value.length != 5)
    CPValido=false
  else{
    for (i=0;i<5;i++){
      CActual = document.f1.codigo.value.charAt(i)
      if (validarEntero(CActual)==""){
        CPValido=false
        break;
      }
    }
  }
  if (!CPValido){
    if (!avisado){
      //si no es valido, Lo aviso
      alert ("Debe escribir un código postal válido")
```

```
        //selecciono el texto
        document.f1.codigo.select()
        //coloco otra vez el foco
        //document.f1.codigo.focus()
        avisado=true
        setTimeout('avisado=false',50)
      }
    }
}
</script>

</head>
<body>

<form name=f1>
Escriba un número entero: <input type=text name=numero size=8 value="" onblur="compruebaValidoEntero()">
<br>
Escriba un código postal: <input type=text name=codigo size=8 value="" onblur="compruebaValidoCP()"> *espera una cadena con 5 carácteres numéricos

</form>

</body>
</html>
```

Este ejemplo sigue la guía del primer ejemplo de onblur de este artículo, incluyendo un nuevo campo a validar.

Para solucionar el tema del bucle infinito, que habréis podido investigar por vosotros mismos y en el que se mostraban una caja de alerta tras otra indefinidamente, hemos utilizado una variable llamada avisado que contiene un true si ya se ha avisado de que el campo estaba mal y un false si no se ha avisado todavía.

Cuando se va a mostrar la caja de alerta se comprueba si se ha avisado o no al usuario. Si ya se avisó no se hace nada, evitando que se muestren más cajas de alerta. Si no se había avisado todavía se muestra la caja de alerta y se coloca el foco en el campo que era incorrecto.

Para restituir la variable avisado a false, de modo que la próxima vez que se escriba mal el valor se muestre el mensaje correspondiente, se utiliza el método setTimeout, que ejecuta la instrucción con un retardo, en este caso de 50 milisegundos. Lo suficiente para que no se meta en un bucle infinito.

> Nota: Después de todos los parches que hemos tenido que colocar para que este evento se comporte correctamente para cumplir el cometido deseado, es posible que no merezca la pena utilizarlo para este cometido. Podemos hacer uso del evento onchange, o comprobar todos los campos de una sola vez cuando el usuario ha decidido enviarlo.

Podemos ver en marcha este ejemplo en una página aparte.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *25/08/2003*
Disponible online en *https://desarrolloweb.com/articulos/1252.php*

Elementos de formulario select asociados

Ejemplo de selects combinados, usando solamente Javascript, de modo que al cambiar un select cambie cambie otro elemento select de la página.

Vamos a conocer uno de los trucos más solicitados de Javascript, que tiene mucha relación con el tema de formularios y donde se utiliza el evento onchange de Javascript. Es un ejemplo sobre cómo realizar una página con un par de selects donde, según el valor escogido en uno de ellos, cambien las opciones posibles del otro select.

Una vez pongas el ejemplo en funcionamiento podrás cambiar la selección del primer select y comprobarás como las opciones del segundo select cambian automáticamente.

El ejemplo que hemos ilustrado utiliza provincias y países. Al escoger en el primer select un país, en el segundo debe mostrarnos las provincias de ese país para que escojamos una provincia, pero sólo una que tenga que esté en el país seleccionado en primer término.

Conocer el objeto select y los option

Es importante conocer los objetos de formulario select y los option. Los select corresponden con las cajas de selección desplegables y los option con cada una de las opciones de la caja desplegable. Podemos ver un artículo que habla de ello.

En concreto nos interesa hacer varias cosas que tienen que ver con extraer el valor de un select en cualquier momento, fijar su número de opciones y, para cada opción, colocar su valor y su texto asociado. Todo esto aprenderemos a hacerlo en este ejemplo.

> Referencia: Para conocer el trabajo con formularios y la jerarquía de objetos javascript (Todo eso es la base del trabajo con los elementos de las páginas en Javascript) debemos haber leer el manual de Javascript II.

Modo de llevar a cabo el problema

Para empezar, vamos a utilizar un formulario con dos selects, uno para el país y otro para la provincia.

```
<form name="f1">
<select name=pais onchange="cambia_provincia()">
<option value="0" selected>Seleccione...
<option value="1">España
<option value="2">Argentina
<option value="3">Colombia
<option value="4">Francia
</select>

<select name=provincia>
<option value="-">-
</select>
</form>
```

Nos fijamos en el select asociado al país de este formulario que, cuando se cambia la opción de país, se debe llamar a la función cambia_provincia(). Veremos más adelante esta función, ahora es importante fijarse que está asociada al evento onchange que se activa cuando cambia la opción en el select.

Todo lo demás será código Javascript. Empezamos definiendo un montón de arrays con las provincias de cada país. En este caso tenemos sólo 4 países, entonces necesitaré 4 arrays. En cada array tengo la lista de provincias de cada país, colocada en cada uno de los elementos del array. Además, dejaré una primera casilla con un valor "-" que indica que no se ha seleccionado ninguna provincia.

```
var provincias_1=new Array("-","Andalucía","Asturias","Baleares","Canarias","Castilla y León","Castilla-La Mancha","...")
var provincias_2=new Array("-","Salta","San Juan","San Luis","La Rioja","La Pampa","...")
var provincias_3=new Array("-","Cali","Santamarta","Medellin","Cartagena","...")
var provincias_4=new Array("-","Aisne","Creuse","Dordogne","Essonne","Gironde ","...")
```

Hay que fijarse que los índices del array de cada país se corresponden con los del select del país. Por ejemplo, la opción España, tiene el valor asociado 1 y el array con las provincias de España se llama provincias_1.

> Actualizado: Observarás que en este ejemplo se usa el método antiguo de creación de arrays en Javascrip, por medio de un constructor Array. Hoy afortunadamente es posible crear arrays en Javascript usando un literal, simplemente colocando sus valores entre corchetes. Luego tenemos un código para que puedas ver esto.

Estos arrays están sueltos, pero generalmente para procesarlos mejor nos conviene tenerlos en un array de dos dimensiones, lo que sería un array de array. Esto es importante para que luego el código para generar los combos dependientes nos salga más sencillo.

Podemos combinar los arrays en uno solo, con este literal de array.

```
var todasProvincias = [
    [],
    provincias_1,
    provincias_2,
    provincias_3,
    provincias_4,
];
```

El script se completa con una función que realiza la carga de las provincias en el segundo select. El mecanismo realiza básicamente estas acciones:

- Detecto el país que se ha seleccionado
- Si el valor del país no es 0 (el valor 0 es cuando no se ha seleccionado país)
    - Tomo el array de provincias adecuado, utilizando el índice del país.
    - Marco el número de opciones que debe tener el select de provincias
    - Para cada opcion del select, coloco su valor y texto asociado, que se hace

corresponder con lo indicado en el array de provincias.
- SI NO (El valor de país es 0, no se ha seleccionado país)
    - Coloco en el select de provincia un único option con el valor "-", que significaba que no había provincia.
- Coloco la opción primera del select de provincia como la seleccionada.

La función tiene el siguiente código. Está comentado para que se pueda entender mejor.

```
function cambia_provincia(){
    //tomo el valor del select del pais elegido
    var pais
    pais = document.f1.pais[document.f1.pais.selectedIndex].value
    //miro a ver si el pais está definido
    if (pais != 0) {
      //si estaba definido, entonces coloco las opciones de la provincia correspondiente.
      //selecciono el array de provincia adecuado
      mis_provincias=todasProvincias[pais]
      //calculo el numero de provincias
      num_provincias = mis_provincias.length
      //marco el número de provincias en el select
      document.f1.provincia.length = num_provincias
      //para cada provincia del array, la introduzco en el select
      for(i=0;i<num_provincias;i++){
        document.f1.provincia.options[i].value=mis_provincias[i]
        document.f1.provincia.options[i].text=mis_provincias[i]
      }
    }else{
      //si no había provincia seleccionada, elimino las provincias del select
      document.f1.provincia.length = 1
      //coloco un guión en la única opción que he dejado
      document.f1.provincia.options[0].value = "-"
      document.f1.provincia.options[0].text = "-"
    }
    //marco como seleccionada la opción primera de provincia
    document.f1.provincia.options[0].selected = true
}
```

Espero que el ejemplo haya sido suficientemente claro para que puedas crear tu sistema de combos dependientes con Javascript, de manera sencilla y rápida.

Código completo de los selects dependientes

Para que te sea más sencillo de comprobar el funcionamiento de este ejemplo y reproducir los pasos para llegar a poner en marcha el ejercicio de los selects dependientes, te pasamos ahora el código completo.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

  <form name="f1">
    <select name=pais onchange="cambia_provincia()">
    <option value="0" selected>Seleccione...
    <option value="1">España
```

```
    <option value="2">Argentina
    <option value="3">Colombia
    <option value="4">Francia
    </select>

    <select name=provincia>
    <option value="-">-
    </select>
  </form>

  <script>
  var provincias_1=new Array("-","Andalucía","Asturias","Baleares","Canarias","Castilla y León","Castilla-La Mancha","...");
  var provincias_2=new Array("-","Salta","San Juan","San Luis","La Rioja","La Pampa","...");
  var provincias_3=new Array("-","Cali","Santamarta","Medellin","Cartagena","...");
  var provincias_4=new Array("-","Aisne","Creuse","Dordogne","Essonne","Gironde ","...");

  var todasProvincias = [
    [],
    provincias_1,
    provincias_2,
    provincias_3,
    provincias_4,
  ];

  function cambia_provincia(){
      //tomo el valor del select del pais elegido
      var pais
      pais = document.f1.pais[document.f1.pais.selectedIndex].value
      //miro a ver si el pais está definido
      if (pais != 0) {
          //si estaba definido, entonces coloco las opciones de la provincia correspondiente.
          //selecciono el array de provincia adecuado
          mis_provincias=todasProvincias[pais]
          //calculo el numero de provincias
          num_provincias = mis_provincias.length
          //marco el número de provincias en el select
          document.f1.provincia.length = num_provincias
          //para cada provincia del array, la introduzco en el select
          for(i=0;i<num_provincias;i++){
              document.f1.provincia.options[i].value=mis_provincias[i]
              document.f1.provincia.options[i].text=mis_provincias[i]
          }
      }else{
          //si no había provincia seleccionada, elimino las provincias del select
          document.f1.provincia.length = 1
          //coloco un guión en la única opción que he dejado
          document.f1.provincia.options[0].value = "-"
          document.f1.provincia.options[0].text = "-"
      }
      //marco como seleccionada la opción primera de provincia
      document.f1.provincia.options[0].selected = true
  }

  </script>
</body>
</html>
```

Verás que funciona perfectamente y que consigues tu objetivo de tener dos campos select donde los valores posibles del segundo campo dependan de la selección realizada en el primero.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *11/10/2019*
Disponible online en *https://desarrolloweb.com/articulos/1281.php*

Evento onunload de Javascript

Ejemplo de uso del evento onunload en Javascript para abrir una ventana secundaria cuando el usuario abandone la página.

Veamos un ejemplo del evento onunload, que, recordamos, se activa cuando el usuario ha abandona la página web. Por tanto, onunload sirve para ejecutar una acción cuando el usuario se marcha de la página, ya sea porque pulsa un enlace que le lleva fuera de la página o porque cierra la ventana del navegador.

El ejemplo que deseamos mostrar sirve para abrir una página web en otra ventana cuando el usuario abandona la página. De este modo actúan muchos de los molestos popups de las páginas web, abriéndose justo cuando abandonamos el sitio que estábamos visitando.

```
<html>
<head>
    <title>Abre al salir</title>
    <script>
    function abreventana(){
      window.open("http://www.google.es","venta","")
    }
    </script>
</head>

<body onunload="abreventana()">

<a href="http://www.desarrolloweb.com">DW!!</a>
</body>
</html>
```

El ejemplo es tan sencillo que casi sobran las explicaciones. Simplemente creamos una función que abre una ventana secundaria y la asociamos con el evento onunload, que se coloca en la etiqueta <body>.

Se puede ver en marcha en una página aparte.

Referencia: Si no tenemos una base de Javascript nos vendrá muy bien acceder a nuestra sección Javascript a fondo.

Si deseamos conocer más cosas de los eventos.

Si deseamos saber más sobre abrir ventanas.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *09/10/2003*
Disponible online en *https://desarrolloweb.com/articulos/1292.php*

Evento onload de Javascript

Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página.

El evento onload de Javascript se activa cuando se termina de cargar la página. Cuando necesitamos hacer cosas en el momento en el que la página haya terminado de cargar, podemos asociar un manejador de evento onload en la etiqueta <body>.

En versiones actuales de Javascript el evento onload también lo aceptan otros elementos de la página, como las imágenes. Como te podrás imaginar, si asociamos un manejador de evento al onload de una imagen, se ejecutará cuando la imagen haya terminado de cargar.

Evento onload para detectar cuándo la página termino de cargar

Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página. Es un evento bastante utilizado pues es muy habitual que se deseen llevar a cabo acciones en ese preciso instante. En nuestro ejemplo vamos a ver cómo podríamos hacer un script para motivar a nuestros visitantes a que nos voten en un ranking cualquiera de páginas web.

La idea es que la página se cargue entera y, una vez esté cargada, aparezca una ventana de Javascript donde se proponga al visitante votar a la página. Es interesante esperar a que termine de cargar la página entera para que el visitante pueda ver la web que se propone votar, antes de que realmente le pidamos su voto.

El código sería el siguiente:

```
<html>
<head>
    <title>Votame!!</title>
<script language="JavaScript">
function pedirVoto(){
    if (confirm("Esta página está genial (ya la puedes ver). Me das tu voto?")){
      window.open("http://www.loquesea.com/votar.php?idvoto=12111","","")
    }
}
</script>
</head>

<body onload="pedirVoto()">
```

```
<h1>Página SuperChula</h1>
<br>
Esta página está muy bonita. Soy su autor y te puedo asegurar que no hay muchas páginas con tanta calidad en Internet
<br>
<br>
<a href="#">Entrar</a>

</body>
</html>
```

Nos fijamos que en la etiqueta <body> tenemos definido un manejador del evento en onload="pedirVoto()". Es decir, que cuando se cargue la página se llamará a una función llamada pedirVoto(), que hemos definido en el bloque de script que tenemos en la cabecera.

La función pedirVoto() utiliza una caja confirm para saber si realmente el usuario desea votarnos. La función confirm() muestra una caja con un texto y dos botones, para aceptar o cancelar. El texto es lo que se recibe por parámetro. Dependiendo de lo que se pulse en los botones, la función confirm() devolverá un true, si se apretó el botón aceptar, o un false, en caso de que se pulsase sobre cancelar.

La función confirm() está a su vez metida dentro de un bloque condicional if, de modo que, dependiendo de lo que se pulsó en la caja confirm, el if se evaluará como positivo o negativo. En este caso sólo se realizan acciones si se pulsó sobre aceptar.

Para acceder a la página donde se contabiliza nuestro voto tenemos el método window.open(), que sirve para abrir ventanas secundarias o popups. Mostramos la página donde se vota en una ventana secundaria para que el visitante no se marche de nuestra web, ya que acaba de entrar y no deseamos que se vaya ya.

Con esto queda más o menos ilustrado cómo hacer uso del evento onload. Seguro que en vuestras creaciones habrá muchos más casos donde utilizarlo.

Definición de manejadores de eventos mediante Javascript

En la actualidad es más común que asociemos los manejadores de eventos directamente con código Javascript, sin usar los atributos del HTML como "onload". Es una buena práctica hacerlo desde Javascript porque así evitamos mezcar código de programación dentro del código HTML, lo que impactará positivamente en la mantenibilidad de la página.

Podemos indicar el manejador de evento "load" sobre el objeto window, de modo que se ejecute el código que necesitemos al cargarse la página. Lo haremos de esta manera:

```
window.addEventListener('load', function() {
    console.log('La página ha terminado de cargarse!!');
});
```

Este código lo podríamos colocar en cualquier lugar de la página, o en un archivo .js externo que se cargue mediante la etiqueta <script>.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *28/02/2022*
Disponible online en *https://desarrolloweb.com/articulos/1319.php*

Eventos personalizados en Javascript

Explicamos qué son los eventos personalizados, cómo se disparan mediante Javascript y cómo asociar manejadores. Aprenderás cómo los eventos escalan mediante la burbuja de eventos y cómo enviar datos a los manejadores.

Los eventos son una de las herramientas más importantes en Javascript para el desarrollo de aplicaciones web. Nos sirven para responder a la interacción del usuario, para posibilitar la interoperabilidad entre plugins o componentes, así como para reaccionar ante cualquier suceso que ocurra en general. Seguro que has trabajado con eventos en Javascript y sabes de lo que estamos hablando, pero ¿Qué sabes de los eventos personalizados?

Los eventos personalizados son una de las herramientas que tenemos para conseguir que los componentes, piezas, plugins o como los quieras llamar se comuniquen entre sí y entre otros elementos de la página. Se usan mucho en el desarrollo de Javascript, por lo que te resultarán muy útiles. En este artículo te explicaremos todo lo que necesitas saber sobre los eventos personalizados, es decir, cómo dispararlos, cómo capturarlos y ejecutar manejadores de eventos, cómo enviar datos junto con los eventos y mucho más.

Para qué sirven los eventos personalizados

La primera vez que oí hablar de los eventos personalizados no entendí muy bien para qué los queríamos. Supongo que podrá pasar parecido con algunos lectores de este artículo.

Los eventos personalizados los entendemos en el contexto de los componentes, por llamar así a cualquier pieza de software que se ejecuta en el navegador. Estos componentes son generalmente piezas que son capaces de encapsular una funcionalidad y permitir su reutilización a lo largo de un proyecto o varios proyectos.

> Dependiendo de tus costumbres o incluso de las tecnologías o librerías con las que trabajes podrás usar el nombre de "componentes", pero puede que los llames "plugins", "custom

elements" de web components o simplemente elementos del DOM que hayas extendido por cualquier medio.

Los componentes generalmente tienen una funcionalidad y cuando pasan cosas con ellos a veces tienen que avisar a otros elementos de su alrededor para informarles que en ellos ha ocurrido algo. Para conseguir eso, en Javascript en el contexto del navegador, levantamos eventos.

Los eventos son señales que nos permiten saber que han pasado cosas. Y "cosas" puede ser algo de lo más amplio. Hay "cosas estándar", como por ejemplo que se haya hecho clic encima de un botón o enlace. Esas "cosas estándar" ya vienen definidas con eventos existentes en Javascript, como "click", pero también hay cosas que pueden ocurrir y que son totalmente arbitrarias, como por ejemplo que hayan pasado 3 minutos desde que se entró en una página web, que se hayan eliminado todos los elementos de una lista, o que se haya seleccionado una fecha del mes siguiente en un calendario.

Esas cosas, por supuesto, no tienen eventos estándar del navegador, pues sería imposible tener eventos ya predefinidos para millones de situaciones que pueden ocurrir en una aplicación web. Para ellas usamos eventos personalizados.

Los eventos personalizados pueden tener cualquier nombre que deseemos nosotros y sirven para informar al exterior de cosas que han ocurrido en los componentes. Espero que más o menos el concepto lo podamos entender.

Por ahondar en el ejemplo del componente de calendario, lo lógico es que el desarrollo sea capaz de adaptarse a la mayor cantidad de situaciones. Por ello sería ideal que el componente avisase de una serie de cosas que puedan ocurrir con él, para que todas las personas que implementen ese calendario lo puedan usar en la mayor cantidad de situaciones. Por ejemplo, sería lógico que este componente avisase con un evento cuando se ha seleccionado una fecha. Avisase si se ha editado a mano la fecha y el formato no es correcto y cosas similares. De este modo, cualquier persona será capaz de usar ese calendario y personalizar comportamientos ya específicos de la aplicación concreta donde se usa el calendario. Por ejemplo podríamos capturar el evento personalizado de cambio de fecha y realizar acciones cuando han seleccionado una fecha del pasado, para informar al usuario que esa fecha no es posible usarla. Vale cualquier ejemplo, ya que podríamos pensar en miles de situaciones en las que diversas partes de la página deben estar atentas a cosas que ocurran en los componentes que estamos creando.

Cómo se dispara un evento personalizado en Javascript

Para disparar eventos personalizados en Javascript nativo, es decir, el Javascript que entiende el navegador, todos los elementos del DOM tienen un método llamado dispatchEvent(). Este es el método que usaremos para levantar nuestros propios eventos personalizados, el cual necesitará que le informemos de los siguientes parámetros:

- El nombre del evento personalizado que queremos disparar, que podría ser cualquiera que nosotros nos inventemos.

- Opcionalmente, se indicará un objeto de configuración del evento.

El código para lanzar un evento personalizado sería más o menos así

```
// Accedo a un elemento del DOM
var elemento = document.getElementById('elemento');
// Disparo el evento personalizado
this.dispatchEvent(new CustomEvent('nombre-del-evento'));
```

Cómo se asocia un manejador de evento personalizado

Los eventos no tendrían mucho sentido si no podemos asociar manejadores de eventos cuando ejecutar acciones cuando éstos se disparan. Los eventos personalizados los capturamos y asociamos manejadores igual que lo hacemos con los eventos normales.

Para ello usamos el método addEventListener(), indicando el nombre del evento personalizado que queremos usar.

```
// Traigo un elemento del DOM
var elemento = document.getElementById('elemento');
// Le asocio un manjeador de evento
lista.addEventListener('nombre-del-evento', function() {
    console.log('detectado evento nombre-del-evento');
});
```

Como puedes ver, los manejadores de eventos se asocian a los elementos del DOM. Dicho de otra manera, los eventos se escuchan sobre los elementos del DOM donde queramos reaccionar a ellos. Este evento lo estamos escuchando sobre el elemento con id="elemento".

Para entender este punto y la importancia que tiene en la gestión de eventos es importante hablar de la burbuja de eventos del navegador.

Burbuja de eventos Javascript

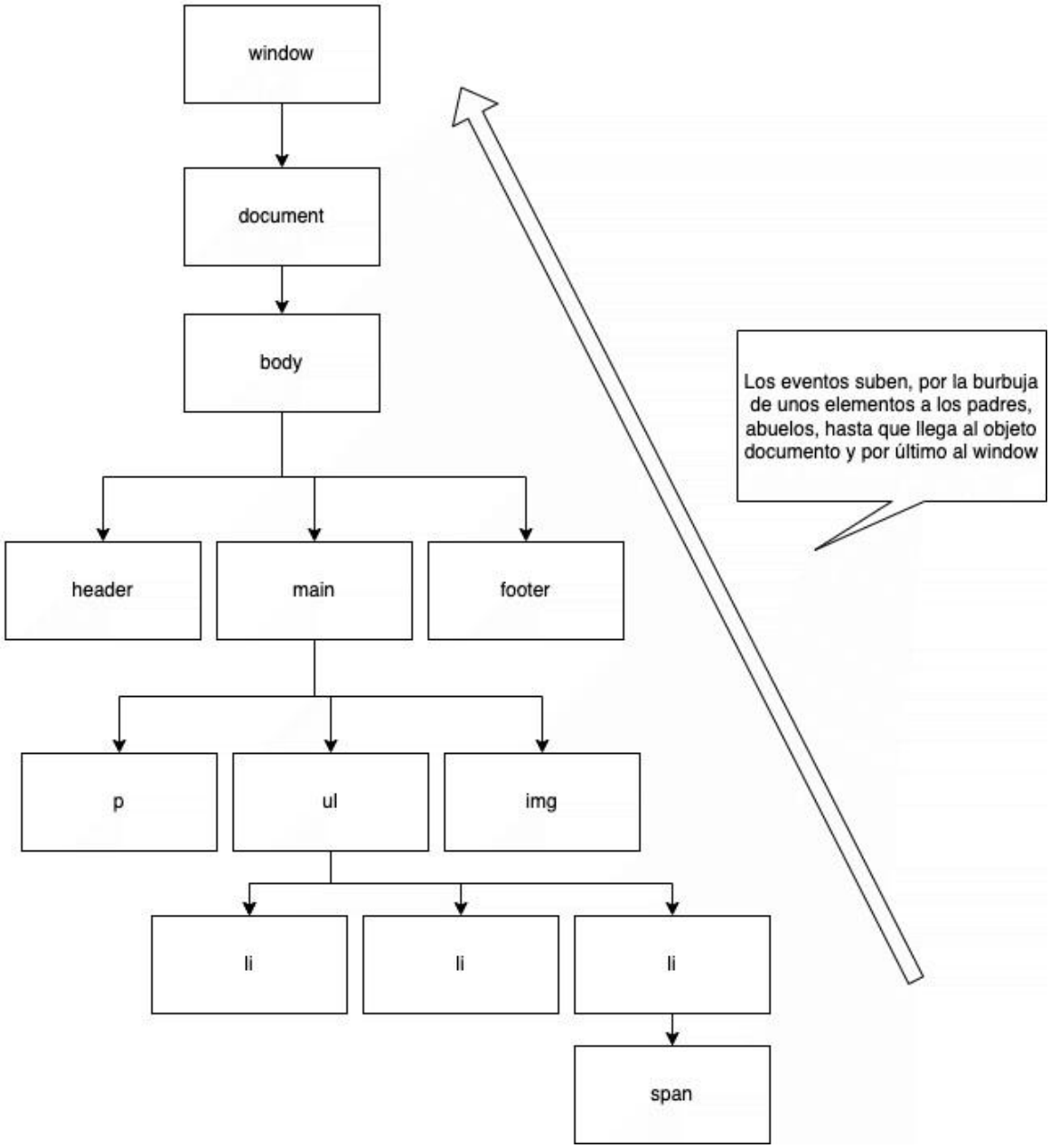
Ten en cuenta que los elementos de la página forman una jerarquía. Esta jerarquía se refiere a la estructura de las etiquetas que hay en el documento HTML y está modelada en lo que llamamos el DOM (Document Object Model).

> Puedes ver el artículo Jerarquía de objetos del navegador para entender mejor este concepto.

La burbuja de eventos del navegador hace que los eventos suban por la jerarquía de objetos, paso a paso, hasta llegar al objeto principal de esta jerarquía, que es el objeto window del navegador.

Esto quiere decir que los eventos irán subiendo, comenzando en el elemento que ha lanzado el evento, subiendo primero a su padre y luego al elemento abuelo, llegando hasta el objeto document y por último al objeto window

Vamos a explicarlo con un ejemplo para que se pueda entender perfectamente. Pero primero veamos la siguiente imagen:



Un evento se puede originar desde cualquier elemento del DOM. Supongamos que del span, que está debajo del todo (aunque podría haber sido en el párrafo, etiqueta "p", o en el header, por ejemplo). Entonces, dada la burbuja y en el caso que el evento se origine sobre el span, este evento lo podríamos escuchar en el propio span, en el elemento li padre, en el elemento ul, el main, body, document y por último en window.

Esto quiere decir que en la práctica podríamos escuchar un evento personalizado originado en el span en todos los elementos de la jerarquía por los que vamos pasando hasta que llegamos al

objeto window.

Pero ojo!! esto no tiene por qué ser siempre así, pues para que el evento personalizado escale mediante la burbuja, tenemos que configurarlo correctamente.

Propiedad bubbles de los eventos personalizados

Los eventos personalizados por defecto no escalan por la burbuja de eventos. Esto quiere decir que solamente los podríamos escuchar con un manejador de eventos que se haya asociado sobre el mismo elemento donde se originó.

Para indicar que el evento personalizado debe subir por la burbuja debemos indicarlo en el objeto de configuración de creación del evento. En ese objeto usaremos una propiedad llamada "bubbles", que debemos asignar a true.

```
elemento.dispatchEvent(new CustomEvent('nombre-del-evento', {
    bubbles: true
}));
```

Con esta sencilla configuración nos aseguramos que el evento subirá hasta llegar a window. Es decir, la burbuja activada hará el trabajo de subir el evento hasta el inicio de la jerarquía.

Por tanto, gracias a la burbuja, si es que está configurada en el evento personalizado, podríamos perfectamente crear el manejador en el objeto document, o en el window, ya que el evento siempre llegará a propagarse hasta llegar a ellos.

```
document.addEventListener('nombre-del-evento', function(e) {
    console.log('El evento también lo puedo capturar en el objeto document, si se activó bubbles');
});
```

Propiedad composed de los eventos personalizados

No me quería olvidar de mencionar la propiedad "composed", que no es tan conocida y ni siquiera la vamos a necesitar siempre, pero sirve para asegurarnos de que el evento suba desde el shadow DOM de un hipotético custom element hacia el DOM común.

Si no trabajas con el estándar de Web Components no te hará mucha falta esta propiedad, aún así este sería un ejemplo:

```
elemento.dispatchEvent(new CustomEvent('nombre-del-evento', {
    bubbles: true,
    composed: true
}));
```

Cómo pasar datos en los eventos personalizados

También es muy interesante la posibilidad de enviar datos junto con el evento. Con ello podemos conseguir informar de cualquier cosa que se necesite al elemento donde estemos escuchando los eventos personalizados.

Para que se entienda por qué queremos enviar datos vamos a poner un par de ejemplos. Vamos a suponer que tenemos elementos de una lista cada elemento de esta lista tiene un botón para eliminar ese ítem. Cuando se pulsa el botón de borrar se puede informar de qué elemento se había intentado borrar, para que la persona que escuche el evento sepa reaccionar correctamente. Ahora por ejemplo piensa en un reproductor de vídeo. Cuando se hace stop sobre el reproductor puede enviar un evento para avisar que se ha parado la reproducción, en cuyo caso sería interesante enviar un dato sobre el segundo en el que el vídeo se ha detenido. Con nuestro componente de fecha, cuando se seleccione una nueva fecha levantaremos un evento personalizado que indicará la fecha que se ha colocado. Puedes encontrar miles de casos en los que puede ser interesante enviar datos junto con el evento.

Enviar los datos al disparar el evento

Para enviar datos en los eventos personalizados usaremos también el objeto de configuración del evento, mediante una propiedad llamada "detail". Veamos este ejemplo de código.

```
elemento.dispatchEvent(new CustomEvent('nombre-del-evento', {
    bubbles: true,
    composed: true,
    detail: {
        saludo: 'hola',
        repeticiones: 3
    }
}));
```

En la propiedad detail he colocado un objeto, que tiene dos propiedades con dos datos. No estoy obligado a que el detail sea siempre un objeto, también podría ser un dato simple, como un número o una cadena.

```
elemento.dispatchEvent(new CustomEvent('nombre-del-evento', {
    bubbles: true,
    composed: true,
    detail: 'id3'
}));
```

Sin embargo, como buena práctica recomendaría siempre enviar objetos en el detail, aunque solamente necesites informar de un único dato, más que nada para conseguir dos beneficios:

- Que el detalle pueda ser más expresivo, indicando qué significado tienen los datos enviados.
- Pero sobre todo, interesa usar un objeto porque así nos aseguramos que, si mañana necesitamos enviar nuevos datos en el detail, sería solamente añadir otras propiedades al objeto.

Recibir los datos en el manejador del evento

Ahora vamos a ver cómo tendríamos que recibir los datos enviados mediante el evento. Para ello usamos el objeto evento que se recibe en cualquier función que hace las veces de manejador del evento.

> Ese objeto evento es muy útil, ya que no solo nos entrega datos sobre el evento que se ha producido, sino que también aporta cierta funcionalidad sobre el evento capturado. Si no necesitamos el objeto evento para nada no es obligatorio recibirlo como parámetro. Se habla más sobre esto en el artículo de los eventos en Javascript.

Con el siguiente código vemos cómo asociar un manejador de evento y cómo recibir el objeto evento. Dentro de ese objeto disponemos de la propiedad "detail", sobre la que podemos acceder a los datos que nos ha enviado el evento, si es que se envió algo.

```
lista.addEventListener('nombre-del-evento', function(e) {
    console.log(`El saludo recibido es ${e.detail.saludo} con ${e.detail.repeticiones} repeticiones`);
});
```

Conclusión

Hemos visto qué son los eventos personalizados en Javascript y para qué los podríamos usar en el contexto de ejecución del navegador. Hemos visto con ejemplos cómo podemos disparar eventos personalizados y cómo podemos capturarlos y ejecutar manejadores de eventos.

La práctica del trabajo con eventos es muy habitual cuando se desarrollan componentes y es básica para que los componentes puedan interactuar con otros componentes o la página pueda reaccionar al uso que se haga de esos componentes. Esperamos que los conocimientos hayan quedado claros y que puedas usarlos en tu día a día del desarrollo frontend.

> Este artículo es obra de *Miguel Angel Alvarez*
> Fue publicado / actualizado en *04/03/2022*
> Disponible online en *https://desarrolloweb.com/articulos/eventos-personalizados-javascript*

Epílogos a la segunda parte del Manual de Javascript

Diversos artículos con informaciones interesantes para completar la formación y para saber por dónde se puede continuar aprendiendo.

Cláusulas try ... catch: detectar y cazar errores en Javascript

Vemos aspectos básicos sobre la utilización de las cláusulas try y catch para detectar y cazar errores javascript, sin que los errores sean tratados por defecto.

Muchos lenguajes de programación utilizan las cláusulas try ... catch para cazar errores y realizar cosas cuando ocurran, por ello, lo que vamos a comentar aquí para Javascript puede resultar muy familiar a los programadores. Estas cláusulas las podemos utilizar para tratar de ejecutar una porción de código, que sabemos que podría desatar un error en tiempo de ejecución.

Cuando ocurre un error en Javascript, se hace un tratamiento determinado (mostrar el error al usuario, ya sea mediante un mensaje o la consola Javascript, o simplemente mostrar un icono y detener la ejecución de ese código). Nosotros con try ... catch podemos evitar que el error se trate de manera predeterminada y que se realicen unas acciones determinadas ante el error. Además, podemos conseguir que el código se siga ejecutando sin ningún problema.

Con try especificamos una serie de sentencias Javascript que vamos a tratar de ejecutar. Con catch especificamos lo que queremos realizar si es que se ha cazado un error en el bloque try. La sintaxis de utilización es la siguiente:

```
try {
   //intento algo que puede producir un error
}catch(mierror){
   //hago algo cuando el error se ha detectado
}
```

El Bloque try se ejecuta tal cual, hasta que un posible error se ha detectado.

- Si no se detecta un error durante la ejecución del bloque try, el catch se deja del lado y no se realiza.
- En caso que sí se detecte un error en el bloque try, se ejecuta las sentencias que teníamos en el catch.

Si nos fijamos, podemos ver que el bloque catch recibe un dato, que en este caso hemos almacenado en la variable "mierror". Esa variable contiene información sobre el error

detectado, que puede ser útil para realizar el tratamiento al error. Veamos este código:

```
try {
  //intento algo que puede producir un error
  funcion_que_no_existe()
}catch(mierror){
  alert("Error detectado: " + mierror.description)
}
```

Cuando se ejecute el try, se detectará un error, al tratar de ejecutar una función que no existe. Entonces se ejecutará la cláusula catch, mostrando el error que se ha detectado. Si nos fijamos, se muestra una descripción del error detectado mediante mierror.description.

Pero la propiedad description del error sólo existe en Internet Explorer. En Firefox, para mostrar una descripción del error lo hacemos directamente con la variable. Así:

```
try {
  //intento algo que puede producir un error
  funcion_que_no_existe()
}catch(mierror){
  alert("Error detectado: " + mierror)
}
```

Entonces, para hacer un bloque try ... catch como este que funcione en los dos navegadores deberíamos hacer esto:

```
try {
  //intento algo que puede producir un error
  funcion_que_no_existe()
}catch(mierror){
  if (mierror.description){
    alert("Error detectado: " + mierror.description)
  }else{
    alert("Error detectado: " + mierror)
  }
}
```

Lanzar una excepción nosotros mismos

También, dentro de un bloque try, podemos lanzar nosotros mismos una excepción, sin que tenga por qué haberse producido un error. Esto lo hacemos con la sentencia trow.

```
try {
throw "miexcepcion"; //lanza una excepción
}
catch (e) {
//tratamos la excepción
alert(e);
}
```

Este código, válido tanto para Internet Explorer como Firefox, lanza una excepción en el bloque try. Luego, en el bloque catch, se muestra la excepción que se ha detectado.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *08/05/2007*
Disponible online en *https://desarrolloweb.com/articulos/clausulas-try-catch.html*

Hojas de referencia Javascript

Varias chuletas u hojas de referencia de Javascript, Con un listado de métodos funciones y usos de Javascript para rápida consulta.

En DesarrolloWeb.com ofrecemos diversos manuales para aprender Javascript, que se pueden consultar todos a través de la sección Javascript a fondo. En esta sección encontraréis numerosos textos explicativos para aprender a trabajar con el lenguaje, tanto para dar los primeros pasos como realizar operaciones más complicadas.

Nuestros manuales vienen con explicaciones detalladas sobre Javascript, pero siempre es bueno tener a mano unas "chuletas", o más dicho de forma más elegante, hojas de referencia rápida. En éstas se encuentran multitud de nombres de funciones, métodos y distintos usos del lenguaje, que son repetidos a menudo durante la programación.

En Internet se pueden encontrar diversas hojas de referencia muy útiles, completas y bien presentadas, que nos pueden ayudar en nuestro día a día con el lenguaje, si las tenemos impresas en papel y a mano para consulta. Lo bueno es que las ofrecen programadores de manera gratuita, con lo que podemos verlas y si nos parece que pueden ayudarnos, se pueden imprimir libremente.

Os paso algunos enlaces interesantes para encontrar estas chuletas de Javascript:

Javascript Quick Reference Card Chuleta de referencia Javascript escrita en dos páginas, con conceptos bastante básicos sobre estructuras de control, operadores, etc, así como objetos y métodos.

JavaScript and Browser Objects Quick Reference 8 páginas con un listado de todos los objetos del navegador y sus propiedades y métodos. Está sacada del libro "Javascript Bible".

Javascript Cheat Sheet: En una sola página se muestran algunas notas sobre sintaxis, los métodos básicos de Javascript, eventos, los métodos DOM e incluso algunas ayudas sobre Ajax.

JavaScript in one page No se podría imprimir en una sola página, pero muestra una completa tabla sobre Javascript, de la que se podría incluso aprender con las escuetas explicaciones.

Referencia Javascript Otra pedazo página-resumen de Javascript, pero que ocupa varias páginas, con referencia sobre sintaxis, orientación a objetos, así como compatibilidad de distintos componentes de la tecnología con cada navegador.

Expresiones regulares en Javascript Con explicaciones muy rápidas y esquemáticas sobre expresiones regulares.

Jquery Cheat Sheet Una página con una referencia rápida para los que trabajan con Jquery 1.2. Ver también esta otra chuleta a color de Jquery.

Prototype Javascript Library 1.5.0 Una superconcentrada hoja de clases y métodos de Prototype.

A field guide to scriptaculous combination effects Referencia rápida a efectos con Scriptaculous.

Mootools 1,1 Cheat Sheet Hoja de referencia a las clases de Mootools.

Chuleta DOM Todos los métodos del DOM de Javascript con rápidas explicaciones.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *05/05/2008*
Disponible online en *https://desarrolloweb.com/articulos/hojas-referencia-javascript.html*

Por dónde continuar aprendiendo Javascript

Qué otras referencias y manuales existen para seguir con tu aprendizaje de Javascript.

En este artículo pretendemos ofrecer un epílogo al Manual de Javascript II y un repaso a otras referencias que existen en este momento dentro de DesarrolloWeb.com que nos pueden ayudar a seguir aprendiendo el lenguaje de programación Javascript y la manera de utilizarlo para realizar todo tipo de efectos e interactividades en la página.

Si has seguido hasta este punto los manuales de Javascript generales, es decir, el Manual de Programación en Javascript I y su segunda parte, el Manual de Programación en Javascript II, estamos seguros que tendrás ya un conocimiento sólido sobre este lenguaje y las posibilidades básicas de la programación en el cliente.

Sin embargo, tú mismo te puedes preguntar cómo hacer muchas de las cosas que ves en tantas y tantas páginas web, como interfaces de usuario avanzadas que responden a la interacción con el visitante, efectos especiales, Ajax, etc. Como podrás comprobar a continuación, todo esto se puede aprender sin salir de DesarrolloWeb.com y afortunadamente ahora está a tu alcance con poco esfuerzo adicional.

Javascript a fondo

La primera referencia que te queremos comentar es la sección dedicada exclusivamente a Javascript dentro de DesarrolloWeb.com. Esto sería como una "portada" del sitio dedicada a Javascript, donde publicamos todos aquellos contenidos que tienen que ver con este lenguaje.

http://www.desarrolloweb.com/javascript/

Taller de Javascript

En el taller de Javascript encontrarás diversos artículos prácticos sobre cómo hacer las cosas más variadas con Javascript. Puedes tratarlos como prácticas, para aprender a hacer una gama de utilidades que te darán una base adicional sobre el lenguaje, o puedes consultarlos cuando tengas que resolver un problema concreto con Javascript.

http://www.desarrolloweb.com/manuales/22/

Otros manuales prácticos

Tenemos también diversos manuales eminentemente prácticos, sobre aspectos muy concretos que se utilizan habitualmente en Javascript. En estos manuales detallamos cosas como el trabajo con imágenes, formularios, frames, ventanas secundarias, etc. Muchas de estas cosas ya las empezamos a tratar en el presente manual, pero existe una información mucho más completa que quizás te interese leer llegado el momento.

Frameworks Javascript

Los frameworks son como librerías de código para hacer tareas comunes en páginas web, creadas por otros programadores y que están a tu disposición para acelerar el proceso de creación de páginas realmente avanzadas. Digamos que cualquiera de nosotros podría programar a mano y desde cero cualquiera de las funcionalidades implementadas en los frameworks, pero ello le ocuparía mucho más tiempo y los resultados lo más probable es que fueran peores.

Los frameworks son sin duda el paso que diferencia el programador de Javascript básico y el programador profesional, sin límites más allá de los propios del navegador y su propia imaginación. Por ello, como te podrás imaginar, recomendamos encarecidamente que aprendas uno de ellos para realizar aplicaciones web basadas en Javascript realmente profesionales.

El único problema que podrás encontrar es que utilizar un framework en muchas de las ocasiones no es una tarea trivial y requiere que el programador tenga bastante habilidad. Sin embargo, en DesarrolloWeb.com hemos tratado en profundidad algunos de los frameworks Javascript más populares y estamos seguros que con nuestros manuales podrás aprender todo lo que necesitas saber para cumplir tus objetivos... y más.

Además, los frameworks solucionan uno de los problemas más grandes que tiene Javascript (si no el más crucial) y es el hecho de que el lenguaje no es exactamente igual en los diversos navegadores del mercado y las versiones que han ido saliendo. Por ello, a medida que se complican las cosas, comprobarás que hay muchas tareas para las cuales tienes que detectar el navegador que está ejecutando la página y realizar acciones distintas dependiendo de ello. Con los frameworks este problema se resuelve y nunca más tendrás que preocuparte de que tu código se ejecute bien en todos los navegadores.

Hablando de librerías Javascript, debemos saber que existen muchas posibilidades que relatamos en el artículo Listado de los Distintos Frameworks Javascript. En DesarrolloWeb.com empezamos hace tiempo a explicar algunos de ellos y comenzamos por uno llamado Cross-browser, pero que realmente no recomendamos, porque se quedó poco

actualizado. Podemos tenerlo en cuenta como un precursor de los frameworks Javascript, pero que se ha quedado en desuso. Actualmente existen opciones mucho mejores.

Sin lugar a dudas, jQuery es el framework Javascript más popular en estos momentos. A poco que se aprenda de jQuery se podrá comprobar que es una auténtica delicia para implementar tanto interactividad como efectos especiales sobre páginas web. Para aprender este framework tenemos un completo Manual de jQuery que no exageramos si decimos que es uno de los mejores manuales ya publicados en DesarrolloWeb.com. Además, hemos publicado también un Taller de jQuery para aprender por la práctica.

Otro de los frameworks populares es Mootools, casi tan bueno como jQuery, pero con sus particularidades. En mi opinión, resulta un poco más complicado de manejar para ciertos usos, pero es una gran librería. Para aprender puedes consultar el Manual de Mootools, tratado también con gran detalle y el Taller de Mootools.

También tenemos un tercer framework en discordia, que nos lo ofrece el equipo de Yahoo!. En esta ocasión el manual sólo tiene una ligera presentación y explicaciones para empezar a trabajar, en el Manual de Introducción a YUI.

Videotutoriales de Javascript

Para quien lo prefiera, en el momento de escribir estas líneas estamos publicando varios materiales para aprender Javascript de una manera más visual. Para ello estamos produciendo unos videotutoriales que te explicarán cosas sobre el propio lenguaje y alguno de los frameworks más utilizados. Consulta el Videotutorial de Javascript y el Videotutorial de jQuery.

Conclusión sobre los contenidos didácticos Javascript

En fin, que como se puede comprobar, en DesarrolloWeb.com tienes todo lo que necesitas para seguir aprendiendo mucho Javascript... Confiamos en que, con un poco de esfuerzo por tu parte, puedas aprovechar todos los materiales que venimos publicando, para llegar a dominar este lenguaje y sus diferentes aplicaciones.

En este artículo ni siquiera hemos nombrado todos los manuales existentes sobre Javascript en este sitio, porque son muchos y algunos de ellos son demasiado específicos. Pero en definitiva, para estar al tanto de todos los manuales que tenemos y podamos seguir publicando en el futuro sobre Javascript, te recomendamos echar un vistazo a la sección de Javascript a fondo.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en *06/10/2010*
Disponible online en *https://desarrolloweb.com/articulos/continuar-aprendiendo-javascript.html*

Un vistazo a los nuevos eventos Touch de JAVASCRIPT

Ese es el primero de una serie de artículos donde veremos y conoceremos una referencia sobre manejo de eventos Touch con JAVASCRIPT.

Con la llegada de HTML5 al mundo del desarrollo web, la forma de crear aplicaciones web ha cambiado de manera muy significativa, ahora bien, JAVASCRIPT no es HTML, pero sí es un elemento de gran importancia para todo lo que significa y se ha logrado con la quinta versión del lenguaje de etiquetas, donde los desarrolladores tienen a mano nuevos elementos. Estos son soportados por nuevas o mejoradas APIS JAVASCRIPT. Aunque no podemos olvidar que la versión tres de CSS ha contribuido en gran medida a una nueva forma versátil y adaptable de visualizar por parte de los usuarios y a crear aplicaciones web por parte de los desarrolladores y diseñadores, con una sutileza semántica.

Todo ha cambiado y JAVASCRIPT lo ha hecho, dando un amplio soporte a dispositivos móviles como smartphones y tablets, además de servir como pilar de nuevos y asombrosos elementos HTML como el CANVAS, donde se construye cualquier cantidad de gráficos a través de JAVASCRIPT, pero en estos momentos lo que nos interesa de las nuevas herramientas del popular lenguaje web del lado del cliente, es el soporte que da al manejo de eventos “Touch”, extendiendo las APIS que tienen como tarea manejar todo lo referente a este tema.

En el presente artículo estudiaremos los diferentes eventos Touch que podemos procesar o manejar con JAVASCRIPT, además de algunos otros aspectos que son de gran importancia para desarrollar aplicaciones que requieran manejos de eventos Touch. Es importante decir que èste es el primero de una serie de artículos, que tienen como fin u objetivo brindar un esquema de orientación donde veremos brevemente algunos usos que

Tipos de eventos Touch

Los dispositivos táctiles trabajan sin ningún problema en cualquier aplicación web, sus navegadores móviles hacen que los eventos de ratón como el clic, terminen por convertirse en un evento Touch, por eso, al hacer una aplicación web, ésta funciona aunque sea procesando eventos del ratón. Ahora se han agregado a JAVASCRIPT una interesante gama de eventos Touch, que a pesar de ser solo de tres tipos, podemos hacer una combinación de los mismos, y así se obtienen mejores resultados. Estos tipos de eventos son:

- touchstart: Este se genera al hacer cualquier toque a la pantalla, sin importar su duración o movimientos realizados.
- touchend: Este se ejecuta una vez se deja de tocar la pantalla o el objeto que tiene asignado el manejador de eventos.

- touchmove: Este es ejecutado una vez se desliza o desplaza el dedo el usuario, por encima de la pantalla u objeto que está siendo controlado a través del manejador de eventos.

Son tres eventos, los cuales son muy simples de entender, con ellos obtenemos aplicaciones web móviles, que sean más dadas a estos entornos táctiles.

> Nota: Es importante mencionar que estos tres eventos funcionan en dispositivos Touch, pero se pueden simular con algún que otro software y también son compatibles con el sistema Touch presente en el ratón de los MacBook Air de Apple.

Herramientas adicionales de los eventos Touch

Hasta ahora ya sabemos que contamos con tres tipos de eventos Touch, éstos, a su vez, tienen una serie de elementos que complementan todo el manejo y procesamientos que se generan tras la interacción de un usuario a través de un dispositivo táctil.

Es importante mencionar que cada evento Touch posee una lista de propiedades en común que vendrían siendo el complemento del que se hablaba anteriormente. Hay tres propiedades que están ligadas de forma directa al Touch, que son:

- touches: Es una lista de todos toques que se han generado en la pantalla, tiene poca utilidad y suele ser poco usada.
- targetTouches: Éste guarda una lista de la cantidad del evento que se ha generado en un elemento del DOM.
- changedTouches: Éste guarda una lista de todos cambios que se producen hasta llegar al evento Touch, por ejemplo, en un touchsend puede haber un touchstart y un touchmove.

Hay otro grupo de propiedades encargado de guardar informacion sobre el evento, las cuales son:

- identifier: Un número único que identifica de forma única cada toque generado durante una sesión.
- target: El elemento del DOM en donde se generó el evento.
- client/ page/ screen: Información de la pantalla, relevante sobre acciones que genera el evento.
- radius coordinates and rotationAngle: Describe una aproximación de las elipses generadas.

Procedimiento para asignar un evento Touch

Los eventos Touch de JAVASCRIPT no cambian para nada el esquema que normalmente se usa para crear manejadores en este lenguaje, así que para iniciar, no se está hablando de diferencias a la hora de asignar un manejador de eventos Touch, antes que cambiar los eventos se han extendido, así que ahora podemos asignar a cualquier elemento del DOM uno o varios

de los tres eventos mencionados algunos párrafos anteriores de éste artículo.

Para facilitar un poco la explicación, veamos algo de código sencillo, iniciaremos por obtener la referencia a través de métodos del DOM.

```
//Obtenemos el elemento con el que vamos a trabajar
var elementoTouch= document.getElementById("areaTactil");
```

Esto no es nada desconocido para los entendidos de JAVASCRIPT, pero si no sabes a cerca del tema, puedes revisar los manuales de JAVASCRIPT que se encuentran a disposición en Desarrolloweb.com. Lo que viene a continuación es la asignación del manejador del evento, además de algunos datos sobre el elemento, como las coordenadas donde se ha llevado a cabo el toque de pantalla.

```
//posteriormente asignamos el manejador de eventos lo cual
// se hace de manera convencional.
elementoTouch.addEventListener('touchstart', function(event){
//Comprobamos si hay varios eventos del mismo tipo
if (event.targetTouches.length == 1) {
var touch = event.targetTouches[0];
// con esto solo se procesa UN evento touch
alert(" se ha producido un touchstart en las siguientes cordenas: X " + touch.pageX + " en Y " + touch.pageY);
}

}, false);
```

En este caso se está manejando un solo evento. En próximos artículos veremos cómo manejar varios eventos, capacidad que solo se puede apreciar en dispositivos multi-Touch.

Este artículo es obra de *Dairo Galeano*
Fue publicado / actualizado en *01/10/2012*
Disponible online en *https://desarrolloweb.com/articulos/eventos-touch-javascript.html*

Datos adicionales en el trabajo con Eventos Touch de JavaScript

Damos los últimos retoques en la triología de artículos dedicada a los eventos Touch de JavaScript. Multi Touch y mucho más.

En dos artículos ya nos hemos dado por enterado de cómo proceder ante el manejo de eventos Touch en aplicaciones Web móvil, un mercado que crece cada día. Por tal razón es que estos dispositivos avanzan de forma acelerada, con un crecimientos abrumador en variedad y avances de herramientas con las que cuentan teléfonos inteligentes y tabletas, que hacen uso de pantallas táctiles para la interacción directa con el usuario. La evolución ha llevado a la concepción de dispositivos multi-Touch, capaces de responder a varios eventos al tiempo, esto ha abierto la puerta a una nueva generación de aplicaciones que están dirigidas a estos sofisticados y avanzados dispositivos, y sólo es posible en tabletas y teléfonos inteligentes de última generación, con sistemas operativos que implementen esta tecnología, pues esta no solo

depende del hardware.

Ante este escenario, JavaScript también se preparó para ser capaz de responder a eventos mult-Touch, haciéndolo de una forma muy sencilla, tal como se han venido manejando los eventos Touch en esta serie de artículos, siendo este el tercero, precedido de los artículos "Un vistazo a los nuevos eventos Touch de JavaScript" y "combinando Eventos Touch de JavaScript". En este donde se pretende rematar los apuntes finales dedicados de mi parte a los eventos Touch de JavaScript, veremos cómo hacer objetos arrastrables, haciendo uso de los eventos Touch, además de cómo procesar múltiples eventos Touch, dirigido a dispositivos avanzados, pero si tu terminal no tiene soporte multi-Touch, de igual forma el ejemplo funcionará, solo que con un solo eventos a la vez.

Haciendo un objeto arrastrable gracias a los eventos Touch

Para engrosar un poco los ejemplos que hemos desarrollado en esta serie de artículos que hemos dedicado al estudio de los eventos Touch a través del lenguaje de la web JavaScript, vamos a crear un ejemplo que, aunque muy simple, nos dará a conocer una forma de hacer un objeto arrastrable. A continuación crearemos un elemento div el cual se podrá mover por la pantalla gracias al deslizamiento que haga el usuario. Para hacerlo posible vamos a usar el evento *touchmove*, haciendo muy fácil la implementación del ejemplo, pero bueno veamos el código que usamos para lograr eso:

```
var obj=document.gelElmentById("objArrastrable");
obj.addEventListener('touchmove', function(event){
    if (event.targetTouches.length == 1) {
        var touch = event.targetTouches[0];
        // con esto solo se procesa UN evento touch
        obj.style.left = touch.pageX + 'px';
        obj.style.top = touch.pageY + 'px';
    }
        }, false);
```

Es importante destacar que ese elemento debe tener algunas reglas de estilo CSS, para poder llevar a feliz término el desarrollo del ejemplo en mención. Inicialmente hay que poner la posición en absoluta del objeto, de lo contrario no se moverá por la pantalla, también debe tener en el estilo CSS los atributos que son modificados a través de los métodos del DOM, es decir, haciendo uso de JavaScript. Como se puede apreciar en el ejemplo, por mi parte hice un pequeño estilo con el fin de poder llevar a cabo la ejecución del ejemplo, estilo que comparto con ustedes, aunque no está de más decir que no se hace realmente algo extraordinario.

```
#objArrastrable{
        border: #000 solid 2px;
        position: absolute;
        width:100px;
        height: 100px;
        top: 450px;
        left: 100px;
      }
```

Con este simple estilo, lo único que garantizamos es la ejecución del ejemplo y será decisión tuya si quieres agregar algo más para mejorar de forma visual tus ejemplos o proyectos que hagan uso de los eventos Touch de JavaScript.

Haciendo un canvas dibujable por el usuario de forma multi Touch

Para quienes hayan aprendido programación Java, quizás hayan podio toparse con un ejemplo muy famoso: es un pequeño espacio donde, al hacer un arrastre con el ratón, se iba dibujando un punto. Este ejemplo aparece en un popular libro de programación, donde lo usaban para enseñar el procesamiento de eventos que se pueden generar con el ratón del ordenador. Traemos a colación ese viejo recuerdo, pues ahora haremos algo parecido, pero nosotros no estamos hablando de eventos del ratón de una computadora, sino de eventos Touch de un dispositivo móvil. Tampoco lo programaremos en Java, sino en JavaScript, usando el elemento canvas de HTML5.

> Nota: Es importante mencionar que, para entender el funcionamiento de este ejemplo, es vital saber manejar el elemento canvas de HTML5. Si no tienes idea acerca de esta nueva etiqueta, te invitamos a que indagues en el Manual de canvas, el cual está disponible en DesarrolloWeb.com.

El ejemplo realmente no hace nada diferente a lo explicado en esta serie de artículos, dedicados fielmente al manejo de eventos Touch. El único cambio drástico es que será capaz de responder a múltiples eventos a la vez, para lo cual creamos un código como el siguiente:

```
canvas.addEventListener('touchmove', function(event) {
    for (var i = 0; i < event.touches.length; i++) {
   var touch = event.touches[i];
    ctx.beginPath();
    ctx.arc(touch.pageX, touch.pageY, 20, 0, 2*Math.PI, true);
    ctx.fill();
    ctx.stroke();
    }
}, false);
```

Como se puede apreciar, procesar múltiples eventos es muy sencillo, sólo hay que hacer un ciclo que itere a través de la lista del objeto touches del evento; de esta forma respondemos a todos los eventos touchmove generados en el *canvas*. No está de más recordar que el funcionamiento de este ejemplo no está garantizado en todos los dispositivos. Su uso está garantizado en iPad dos, Motorola Xoom, donde se ha probado el funcionamiento de este ejemplo.

También compartimos el código completo del ejemplo, además de la opción de verlos en funcionamiento en una página aparte.

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Estacando eventos Touch</title>
   <meta charset="utf-8" />
   <script>
     function inicia(){
       var obj=document.getElementById("objArrastrable");

        var canvas = document.getElementById('objeto');
        var ctx= canvas.getContext('2d');
obj.addEventListener('touchmove', function(event){
   if (event.targetTouches.length == 1) {
     var touch = event.targetTouches[0];
     // con esto solo se procesa UN evento touch
     obj.style.left = touch.pageX + 'px';
     obj.style.top = touch.pageY + 'px';
   }
     }, false);

  canvas.addEventListener('touchmove', function(event) {
   for (var i = 0; i < event.touches.length; i++) {
  var touch = event.touches[i];
   ctx.beginPath();
   ctx.arc(touch.pageX, touch.pageY, 20, 0, 2*Math.PI, true);
   ctx.fill();
   ctx.stroke();
   }
}, false);
  }
   </script>

   <style>
     #objArrastrable{
       border: #000 solid 2px;
       position: absolute;
       width:100px;
       height: 100px;
       top: 450px;
       left: 100px;
     }
   </style>
</head>
   <body onLoad="inicia();">
    <canvas width="450" height="350" style="border: #000 solid 3px; " id="objeto"></canvas>
    <div id="objArrastrable">
      Objeto arrastrable
    </div>
   </body>
</html>
```

Este artículo es obra de *Dairo Galeano*
Fue publicado / actualizado en *24/10/2012*
Disponible online en *https://desarrolloweb.com/articulos/trabajo-eventos-touch-javascript.html*