Eventos en JavaScript

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (if, for, while) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes *eventos* que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de *programación basada en eventos*.

En este tipo de programación, los scripts se dedican a esperar a que el usuario *"haga algo"* (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *"event handlers"* en inglés y suelen traducirse por *"manejadores de eventos"*.

Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina *onclick* y el evento asociado a la acción de mover el ratón se denomina *onmousemove*.

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pinchar y soltar el ratón	Todos los elementos
`ondblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`

onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son *onload* para esperar a que se cargue la página por completo, los eventos *onclick*, *onmouseover*, *onmouseout* para controlar el ratón y *onsubmit* para controlar el envío de los formularios.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo *<input type="submit">* se desencadenan los eventos *onmousedown*, *onclick*, *onmouseup* y *onsubmit* de forma consecutiva.

Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *"manejador de eventos"* y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores *"semánticos"*.

Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="console.log('Gracias por pinchar');" />
```

```
<div onclick="console.log('Has pinchado con el ratón');" onmouseover="console.log('Acabas de p
asar el ratón por encima');">
    Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es *onclick*. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado *onclick*.

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas

```
<body onload="console.log('La página se ha cargado completamente');">
 ...
</body>
```

Manejadores de eventos y variable this

JavaScript define una variable especial llamada *this* que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable *this* para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente.

Cuando el usuario pasa el ratón por encima del , el color del borde se muestra de color negro. Cuando el ratón sale del , se vuelve a mostrar el borde con el color gris claro original.

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver" onmouseover="d
ocument.getElementById('contenidos').style.borderColor='black';" onmouseout="document.getEleme
ntById('contenidos').style.borderColor='silver';">
    Sección de contenidos...
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable this, que hace referencia al elemento XHTML que ha provocado el evento.

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver" onmouseover="t
his.style.borderColor='black';" onmouseout="this.style.borderColor='silver';">
    Sección de contenidos...
</div>
```

Manejadores de eventos como funciones externas

La definición de manejadores de eventos en los atributos XHTML es un método sencillo pero poco aconsejable para tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Cuando el código de la función manejadora es más complejo, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa que se invoca desde el código XHTML cuando se produce el evento.

```
<input type="button" value="Pinchame y verás" onclick="console.log('Gracias por pinchar');" />
```

Se puede transformar en:

```
function muestraMensaje() {
  console.log('Gracias por pinchar');
}
```

```
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable *this* y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}
```

```
<div style="padding: .2em; width: 150px; height: 60px; border: thin solid silver" onmouseover=
"resalta(this)" onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

Por otra parte, el ejemplo anterior se complica por la forma en la que los distintos navegadores almacenan el valor de la propiedad *borderColor*. Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor simple *black*, Internet Explorer lo almacena como *black black black black* y Opera almacena su representación hexadecimal *#000000*.

Manejadores de eventos semánticos

Utilizar los atributos XHTML o las funciones externas para añadir manejadores de eventos tiene un grave inconveniente: "ensucian" el código XHTML de la página.

Como es conocido, al crear páginas web se recomienda separar los contenidos (XHTML) de la presentación (CSS). En lo posible, también se recomienda separar los contenidos (XHTML) de la programación (JavaScript). Mezclar JavaScript y XHTML complica excesivamente el código fuente de la página, dificulta su mantenimiento y reduce la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica consiste en asignar las funciones externas mediante las propiedades DOM de los elementos XHTML.

```
function muestraMensaje() {
  console.log('Gracias por pinchar');
}
document.getElementById("pinchable").onclick = muestraMensaje;
```

```
<input id="pinchable" type="button" value="Pinchame y verás" />
```

El código XHTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript. La técnica de los manejadores semánticos consiste en:

- Asignar un identificador único al elemento XHTML mediante el atributo id.
- Crear una función de JavaScript encargada de manejar el evento.
- Asignar la función a un evento concreto del elemento XHTML mediante DOM.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento *onload*:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

El flujo de eventos

Además de los eventos básicos que se han visto, los navegadores incluyen un mecanismo relacionado llamado flujo de eventos o *"event flow"*. El flujo de eventos permite que varios elementos diferentes puedan responder a un mismo evento.

Si en una página HTML se define un elemento con un botón en su interior, cuando el usuario pulsa sobre el botón, el navegador permite asignar una función de respuesta al botón, otra función de respuesta al que lo contiene y otra función de respuesta a la página completa. De esta forma, un solo evento (la pulsación de un botón) provoca la respuesta de tres elementos de la página (incluyendo la propia página).

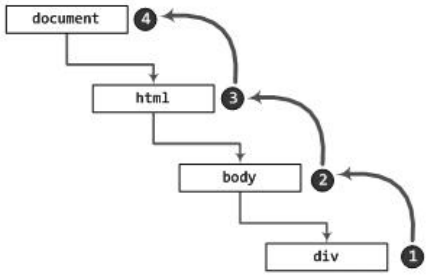
El orden en el que se ejecutan los eventos asignados a cada elemento de la página es lo que constituye el flujo de eventos. Además, existen muchas diferencias en el flujo de eventos de cada navegador.

Event bubbling

En este modelo de flujo de eventos, el orden que se sigue es desde el elemento más específico hasta el elemento menos específico.

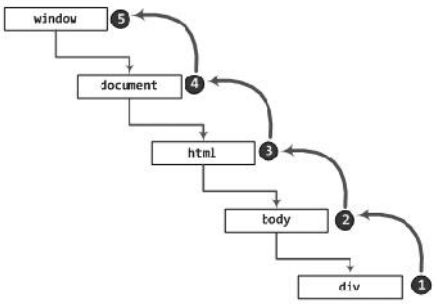
```
<html onclick="procesaEvento()">
  <head><title>Ejemplo de flujo de eventos</title></head>
  <body onclick="procesaEvento()">
    <div onclick="procesaEvento()">Pincha aqui</div>
  </body>
</html>
```

Cuando se pulsa sobre el texto "Pincha aquí" que se encuentra dentro del , se ejecutan los siguientes eventos en el orden que muestra el siguiente esquema:



El primer evento que se tiene en cuenta es el generado por el que contiene el mensaje. A continuación el navegador recorre los ascendentes del elemento hasta que alcanza el nivel superior, que es el elemento *document*.

Este modelo de flujo de eventos es el que incluye el navegador Internet Explorer. Los navegadores de la familia Mozilla (por ejemplo Firefox) también soportan este modelo, pero ligeramente modificado. El anterior ejemplo en un navegador de la familia Mozilla presenta el siguiente flujo de eventos:



Event capturing

En ese otro modelo, el flujo de eventos se define desde el elemento menos específico hasta el elemento más específico. En otras palabras, el mecanismo definido es justamente el contrario al "event bubbling". Este modelo lo utilizaba el desaparecido navegador Netscape Navigator 4.0.

Handlers y listeners

En las secciones anteriores se introdujo el concepto de "*event handler*" o manejador de eventos, que son las funciones que responden a los eventos que se producen.

Cualquiera de los tres modelos anteriores funciona correctamente en todos los navegadores disponibles en la actualidad. Las diferencias entre navegadores surgen cuando se define más de un manejador de eventos para un mismo evento de un elemento. La forma de asignar y *"desasignar"* manejadores múltiples depende completamente del navegador utilizado.

Manejadores de eventos de DOM

La especificación DOM define otros dos métodos similares a los disponibles para Internet Explorer y denominados *addEventListener()* y *removeEventListener()* para asociar y desasociar manejadores de eventos.

La principal diferencia entre estos métodos y los anteriores es que en este caso se requieren tres parámetros:

- el nombre del *"event listener"*
- una referencia a la función encargada de procesar el evento
- el tipo de flujo de eventos al que se aplica.

El primer argumento no es el nombre completo del evento como sucede en el modelo de Internet Explorer, sino que se debe eliminar el prefijo *on*. En otras palabras, si en Internet Explorer se utilizaba el nombre *onclick*, ahora se debe utilizar *click*.

Si el tercer parámetro es *true*, el manejador se emplea en la fase de *capture*. Si el tercer parámetro es *false*, el manejador se asocia a la fase de *bubbling*.

A continuación, se muestran los ejemplos anteriores empleando los métodos definidos por DOM:

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, false);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje, false);
```

Asociando múltiples funciones a un único evento

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}

function muestraOtroMensaje() {
  console.log("Has pulsado el ratón y por eso se muestran estos mensajes");
}

var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, true);
elDiv.addEventListener("click", muestraOtroMensaje, true);
```

Si se asocia una función a un flujo de eventos determinado, esa función sólo se puede desasociar en el mismo tipo de flujo de eventos. Si se considera el siguiente ejemplo:

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}

var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, false);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje, true);
```

El objeto event

Cuando se produce un evento, no es suficiente con asignarle una función responsable de procesar ese evento. Normalmente, la función que procesa el evento necesita información relativa al evento producido: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.

El objeto *event* es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignadas al evento.

El estándar DOM especifica que el objeto *event* es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos. Por tanto, en los navegadores que siguen los estándares, se puede acceder al objeto *event* a través del array de los argumentos de la función:

```
elDiv.onclick = function() {
  var elEvento = arguments[0];
}
```

```
elDiv.onclick = function(event) {
  ...
}
```

El funcionamiento de los navegadores que siguen los estándares puede parecer "mágico", ya que en la declaración de la función se indica que tiene un parámetro, pero en la aplicación no se pasa ningún parámetro a esa función. En realidad, los navegadores que siguen los estándares crean automáticamente ese parámetro y lo pasan siempre a la función encargada de manejar el evento.

Propiedades y métodos

A pesar de que el mecanismo definido por los navegadores para el objeto event es similar, existen numerosas diferencias en cuanto las propiedades y métodos del objeto.

Propiedad/Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `ALT` y `false` en otro caso
`bubbles`	Boolean	Indica si el evento pertenece al flujo de eventos de bubbling
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: `0` Ningún botón pulsado `1` Se ha pulsado el botón izquierdo `2` Se ha pulsado el botón derecho `3` Se pulsan a la vez el botón izquierdo y el derecho `4` Se ha pulsado el botón central `5` Se pulsan a la vez el botón izquierdo y el central `6` Se pulsan a la vez el botón derecho y el central `7` Se pulsan a la vez los 3 botones
`cancelable`	Boolean	Indica si el evento se puede cancelar
`cancelBubble`	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling*
`charCode`	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
`clientX`	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
`clientY`	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
`ctrlKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `CTRL` y `false` en otro caso
`currentTarget`	Element	El elemento que es el objetivo del evento
`detail`	Número entero	El número de veces que se han pulsado los botones del ratón
`eventPhase`	Número entero	La fase a la que pertenece el evento: `0` Fase capturing `1` En el elemento destino `2` Fase bubbling
`isChar`	Boolean	Indica si la tecla pulsada corresponde a un carácter
`keyCode`	Número entero	Indica el código numérico de la tecla pulsada
`metaKey`	Número entero	Devuelve `true` si se ha pulsado la tecla `META` y `false` en otro caso
`pageX`	Número entero	Coordenada X de la posición del ratón respecto de la página
`pageY`	Número entero	Coordenada Y de la posición del ratón respecto de la página
`preventDefault()`	Función	Se emplea para cancelar la acción predefinida del evento
`relatedTarget`	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
`screenX`	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
`screenY`	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
`shiftKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `SHIFT` y `false` en otro caso
`stopPropagation()`	Función	Se emplea para detener el flujo de eventos de tipo bubbling
`target`	Element	El elemento que origina el evento
`timeStamp`	Número	La fecha y hora en la que se ha producido el evento
`type`	Cadena de texto	El nombre del evento

Al contrario de lo que sucede con Internet Explorer, la mayoría de propiedades del objeto *event* de DOM son de sólo lectura. En concreto, solamente las siguientes propiedades son de lectura y escritura: *altKey, button y keyCode*.

Similitudes y diferencias entre navegadores

En ambos casos se utiliza la propiedad type para obtener el tipo de evento que se trata:

```
function procesaEvento(elEvento) {
  if(elEvento.type == "click") {
    console.log("Has pulsado el raton");
  }
  else if(elEvento.type == "mouseover") {
    console.log("Has movido el raton");
  }
}

elDiv.onclick = procesaEvento;
elDiv.onmouseover = procesaEvento;
```

Mientras que el manejador del evento incluye el prefijo *on* en su nombre, el tipo de evento devuelto por la propiedad *type* prescinde de ese prefijo. Por eso en el ejemplo anterior se compara su valor con *click* y *mouseover* y no con *onclick* y *onmouseover*.

Otra similitud es el uso de la propiedad *keyCode* para obtener el código correspondiente al carácter de la tecla que se ha pulsado. La tecla pulsada no siempre representa un carácter alfanumérico. Cuando se pulsa la tecla ENTER por ejemplo, se obtiene el código 13. La barra espaciadora se corresponde con el código 32 y la tecla de borrado tiene un código igual a 8.

Para obtener la posición del ratón respecto de la parte visible de la ventana, se emplean las propiedades *clientX* y *clientY*. De la misma forma, para obtener la posición del puntero del ratón respecto de la pantalla completa, se emplean las propiedades *screenX* y *screenY*.

Una de las principales diferencias es la forma en la que se obtiene el elemento que origina el evento. Si un elemento <*div*> tiene asignado un evento *onclick*, al pulsar con el ratón el interior del <*div*> se origina un evento cuyo objetivo es el elemento <*div*>.

```
// Internet Explorer
var objetivo = elEvento.srcElement;

// Navegadores que siguen los estandares
var objetivo = elEvento.target;
```

Una de las propiedades más interesantes es la posibilidad de impedir que se complete el comportamiento normal de un evento. En otras palabras, con JavaScript es posible no mostrar ningún carácter cuando se pulsa una tecla, no enviar un formulario después de pulsar el botón de envío, no cargar ninguna página al pulsar un enlace, etc. El método avanzado de impedir que un evento ejecute su acción asociada depende de cada navegador.

```
// Navegadores que siguen los estandares
elEvento.preventDefault();
```

En el modelo básico de eventos también es posible impedir el comportamiento por defecto de algunos eventos. Si por ejemplo en un elemento <*textarea*> se indica el siguiente manejador de eventos:

```
<textarea onkeypress="return false;"></textarea>
```

En el <*textarea*> anterior no será posible escribir ningún carácter, ya que el manejador de eventos devuelve false y ese es el valor necesario para impedir que se termine de ejecutar el evento y por tanto para evitar que la letra se escriba.
Así, es posible definir manejadores de eventos que devuelvan true o false en función de algunos parámetros. Por ejemplo se puede diseñar un limitador del número de caracteres que se pueden escribir en un <*textarea*>:

```
function limita(maximoCaracteres) {
  var elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres ) {
    return false;
  }
  else {
    return true;
  }
}
```

```
<textarea id="texto" onkeypress="return limita(100);"></textarea>
```

Tipos de eventos

La lista completa de eventos que se pueden generar en un navegador se puede dividir en cuatro grandes grupos. La especificación de DOM define los siguientes grupos:

- Eventos de **ratón**: se originan cuando el usuario emplea el ratón para realizar algunas acciones.
- Eventos de **teclado**: se originan cuando el usuario pulsa sobre cualquier tecla de su teclado.
- Eventos **HTML**: se originan cuando se producen cambios en la ventana del navegador o cuando se producen ciertas interacciones entre el cliente y el servidor.
- Eventos **DOM**: se originan cuando se produce un cambio en la estructura DOM de la página. También se denominan "eventos de mutación".

EVENTOS DE RATON:

Evento	Descripción
`click`	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla `ENTER`
`dblclick`	Se produce cuando se pulsa dos veces el botón izquierdo del ratón
`mousedown`	Se produce cuando se pulsa cualquier botón del ratón
`mouseout`	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario mueve el puntero a un lugar fuera de ese elemento
`mouseover`	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento
`mouseup`	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado
`mousemove`	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento

EVENTOS DE TECLADO

Evento	Descripción
`keydown`	Se produce cuando se pulsa cualquier tecla del teclado. También se produce de forma continua si se mantiene pulsada la tecla
`keypress`	Se produce cuando se pulsa una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta teclas como `SHIFT`, `ALT`, etc.). También se produce de forma continua si se mantiene pulsada la tecla
`keyup`	Se produce cuando se suelta cualquier tecla pulsada

EVENTOS DE HTML

Evento	Descripción
`load`	Se produce en el objeto window cuando la página se carga por completo. En el elemento `<img>` cuando se carga por completo la imagen. En el elemento `<object>` cuando se carga el objeto
`unload`	Se produce en el objeto window cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo). En el elemento `<object>` cuando desaparece el objeto.
`abort`	Se produce en un elemento `<object>` cuando el usuario detiene la descarga del elemento antes de que haya terminado
`error`	Se produce en el objeto window cuando se produce un error de JavaScript. En el elemento `<img>` cuando la imagen no se ha podido cargar por completo y en el elemento `<object>` cuando el elemento no se carga correctamente
`select`	Se produce cuando se seleccionan varios caracteres de un cuadro de texto (`<input>` y `<textarea>`)
`change`	Se produce cuando un cuadro de texto (`<input>` y `<textarea>`) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento `<select>`
`submit`	Se produce cuando se pulsa sobre un botón de tipo submit (`<input type="submit">`)
`reset`	Se produce cuando se pulsa sobre un botón de tipo reset (`<input type="reset">`)
`resize`	Se produce en el objeto window cuando se redimensiona la ventana del navegador
`scroll`	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento `<body>` contiene la barra de scroll de la página completa
`focus`	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco
`blur`	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco

EVENTOS DE DOM

Evento	Descripción
`DOMSubtreeModified`	Se produce cuando se añaden o eliminan nodos en el subárbol de un documento o elemento
`DOMNodeInserted`	Se produce cuando se añade un nodo como hijo de otro nodo
`DOMNodeRemoved`	Se produce cuando se elimina un nodo que es hijo de otro nodo
`DOMNodeRemovedFromDocument`	Se produce cuando se elimina un nodo del documento
`DOMNodeInsertedIntoDocument`	Se produce cuando se añade un nodo al documento