Eventos

Introducción

- Nos permiten detectar acciones que realiza el usuario o cambios que suceden en la página y reaccionar en respuesta a ellas. Existen muchos eventos diferentes (podéis ver la lista en w3schools) aunque nosotros nos centraremos en los más comunes.
- Javascript nos permite ejecutar código cuando se produce un evento (por ejemplo el evento *click* del ratón) asociando al mismo una función. Hay varias formas de hacerlo.

- **Cómo escuchar un evento**
- La primera manera “estándar” de asociar código a un evento era añadiendo un atributo con el nombre del evento a escuchar (con ‘on’ delante) en el elemento HTML. Por ejemplo, para ejecutar código al producirse el evento ‘click’ sobre un botón se escribía:

```
<input type="button" id="boton1" onclick="alert('Se ha pulsado');" />
```

```
<input type="button" id="boton1" onclick="clicked()" />
```

```
function clicked() {
  alert('Se ha pulsado');
}
```

Esto “ensuciaba” con código la página HTML por lo que se creó el modelo de registro de eventos tradicional que permitía asociar a un elemento HTML una propiedad con el nombre del evento a escuchar (con ‘on’ delante). En el caso anterior:

```
document.getElementById('boton1').onclick = funnction () {
  alert('Se ha pulsado');
}
...
```

NOTA: hay que tener cuidado porque si se ejecuta el código antes de que se haya creado el botón estaremos asociando la función al evento *click* de un elemento que aún no existe así que no hará nada. Para evitarlo siempre es conveniente poner el código que atiende a los eventos dentro de una función que se ejecute al producirse el evento *load* de la ventana. Este evento se produce cuando se han cargado todos los elementos HTML de la página y se ha creado el árbol DOM. Lo mismo habría que hacer con cualquier código que modifique el árbol DOM. El código correcto sería:

```
window.onload = function() {
  document.getElementById('boton1').onclick = function() {
    alert('Se ha pulsado');
  }
}
```

Event listeners

La forma recomendada de hacerlo es usando el modelo avanzado de registro de eventos del W3C. Se usa el método addEventListener que recibe como primer parámetro el nombre del evento a escuchar (sin ‘on’) y como segundo parámetro la función a ejecutar (OJO, sin paréntesis) cuando se produzca:

```
document.getElementById('boton1').addEventListener('click', pulsado);
...
function pulsado() {
  alert('Se ha pulsado');
})
```

Si queremos pasarle algún parámetro a la función escuchadora (cosa bastante poco usual) debemos usar funciones anónimas como escuchadores de eventos:

```
window.addEventListener('load', function() {
  document.getElementById('acepto').addEventListener('click', function() {
    alert('Se ha aceptado');
  })
})
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add many events on the same button.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>
```

Caso: varios eventos asociados a un botón

Ejemplo con verificación de carga

Javascript

```
window.addEventListener('load', function() {
  document.getElementById('acepto').addEventListener('click', function() {
    alert('Se ha aceptado');
  })
})
```

HTML

```
<button id="acepto">Aceptar</button>
```

NOTA: igual que antes debemos estar seguros de que se ha creado el árbol DOM antes de poner un escuchador por lo que se recomienda ponerlos siempre dentro de la función asociada al evento window.onload (o mejor window.addEventListener('load', ...) como en el ejemplo anterior).

Una ventaja de este método es que podemos poner varios escuchadores para el mismo evento y se ejecutarán todos ellos. Para eliminar un escuchador se usa el método removeEventListener.

```
document.getElementById('acepto').removeEventListener('click', aceptado);
```

NOTA: no se puede quitar un escuchador si hemos usado una función anónima, para quitarlo debemos usar como escuchador una función con nombre.

El ciclo de vida de una página HTML tiene tres eventos importantes:

- `DOMContentLoaded` – el navegador HTML está completamente cargado y el árbol DOM está construido, pero es posible que los recursos externos como `<img>` y hojas de estilo aún no se hayan cargado.
- `load` – no solo se cargó el HTML, sino también todos los recursos externos: imágenes, estilos, etc.
- `beforeunload/unload` – el usuario sale de la pagina.

Cada evento puede ser útil:

- Evento `DOMContentLoaded` – DOM está listo, por lo que el controlador puede buscar nodos DOM, inicializar la interfaz.
- Evento `load` – se cargan recursos externos, por lo que se aplican estilos, se conocen tamaños de imagen, etc.
- Evento `beforeunload` – el usuario se va: podemos comprobar si el usuario guardó los cambios y preguntarle si realmente quiere irse.
- Evento `unload` – el usuario casi se fue, pero aún podemos iniciar algunas operaciones, como enviar estadísticas.

DOMContentLoaded

```
<!doctype html>
<body>
<script>
  function ready() {
    alert('DOM is ready');

    // la imagen aún no está cargada (a menos que se haya almacenado en caché),
      por lo que el tamaño es 0x0
    alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);
  }

  document.addEventListener("DOMContentLoaded", ready);
</script>

<img id="img" src="https://en.js.cx/clipart/train.gif?speed=1&cache=0">
</body>
```

En el ejemplo, el controlador del evento DOMContentLoaded se ejecuta cuando el documento está cargado, por lo que puede ver todos los elementos, incluido el <img> que está después de él.

Pero no espera a que se cargue la imagen. Entonces, alert muestra los tamaños en cero.

window.onload
El evento load en el objeto window se activa cuando se carga toda la página, incluidos estilos, imágenes y otros recursos. Este evento está disponible a través de la propiedad onload.

El siguiente ejemplo muestra correctamente los tamaños de las imágenes, porque window.onload espera todas las imágenes:

```
<!doctype html>
<body>
<script>
  window.onload = function() { // también puede usar window.addEventListener
    ('load', (event) => {
    alert('Página cargada');

    // la imagen es cargada al mismo tiempo
    alert(`Tamaño de imagen: ${img.offsetWidth}x${img.offsetHeight}`);
  };
</script>

<img id="img" src="https://en.js.cx/clipart/train.gif?speed=1&cache=0">
</body>
```

readyState
¿Qué sucede si configuramos el controlador DOMContentLoaded después de cargar el documento?

Naturalmente, nunca se ejecutará.

Hay casos en los que no estamos seguros de si el documento está listo o no. Nos gustaría que nuestra función se ejecute cuando se cargue el DOM, ya sea ahora o más tarde.

La propiedad document.readyState nos informa sobre el estado de carga actual.

Hay 3 valores posibles:

"loading" – el documento se está cargando.
"interactive" – el documento fue leído por completo.
"complete" – el documento se leyó por completo y todos los recursos (como imágenes) también se cargaron.

```
<!DOCTYPE html>
<script>
"use strict";

// estado actual
console.log(document.readyState);

//imprimir los cambios de estado
document.addEventListener('readystatechange', () => console.log(document.readyState));
</script>
```

Tipos de eventos

Eventos de página

Se producen en el documento HTML, normalmente en el BODY:

- **load**: se produce cuando termina de cargarse la página (cuando ya está construido el árbol DOM). Es útil para hacer acciones que requieran que el DOM esté cargado como modificar la página o poner escuchadores de eventos
- **unload**: al destruirse el documento (ej. cerrar)
- **beforeUnload**: antes de destruirse (podríamos mostrar un mensaje de confirmación)
- **resize**: si cambia el tamaño del documento (porque se redimensiona la ventana)

Eventos de ratón

Los produce el usuario con el ratón:

- **click** / **dblclick**: cuando se hace click/doble click sobre un elemento
- **mousedown** / **mouseup**: al pulsar/soltar cualquier botón del ratón
- **mouseenter** / **mouseleave**: cuando el puntero del ratón entra/sale del elemento (tb. podemos usar mouseover/mouseout)
- **mousemove**: se produce continuamente mientras el puntero se mueva dentro del elemento

Eventos de toque

Se producen al usar una pantalla táctil:

- **touchstart**: se produce cuando se detecta un toque en la pantalla táctil
- **touchend**: cuando se deja de pulsar la pantalla táctil
- **touchmove**: cuando un dedo es desplazado a través de la pantalla
- **touchcancel**: cuando se interrumpe un evento táctil.

- **focus / blur**: al obtener/perder el foco el elemento
- **change**: al perder el foco un <input> o <textarea> si ha cambiado su contenido o al cambiar de valor un <select> o un <checkbox>
- **input**: al cambiar el valor de un <imput> o <textarea> (se produce cada vez que escribimos una letra es estos elementos)
- **select**: al cambiar el valor de un <select> o al seleccionar texto de un <imput> o <textarea>
- **submit / reset**: al enviar/recargar un formulario

Los objetos *this* y *event*

Al producirse un evento se generan automáticamente en su función manejadora 2 objetos:

- **this**: siempre hace referencia al elemento que contiene el código en donde se encuentra la variable *this*. En el caso de una función escuchadora será el elemento que tiene el escuchador que ha recibido el evento
- **event**: es un objeto y la función escuchadora lo recibe como parámetro. Tiene propiedades y métodos que nos dan información sobre el evento, como:

    - **.preventDefault()**: si un evento tiene un escuchador asociado se ejecuta el código de dicho escuchador y después el navegador realiza la acción que correspondería por defecto al evento si no tuviera escuchador (por ejemplo un escuchador del evento *click* sobre un hiperenlace hará que se ejecute su código y después saltará a la página indicada en el *href* del hiperenlace). Este método cancela la acción por defecto del navegador para el evento. Por ejemplo si el evento era el *submit* de un formulario éste no se enviará o si era un *click* sobre un hiperenlace no se irá a la página indicada en él.

    - **.stopPropagation**: un evento se produce sobre un elemento y todos su padres. Por ejemplo si hacemos click en un que está en un <p> que está en un que está en el BODY el evento se va propagando por todos estos elementos y saltarían los escuchadores asociados a todos ellos (si los hubiera). Si alguno llama a este método el evento no se propagará a los demás elementos padre.

Propagación de eventos (bubbling)

Normalmente en una página web los elementos HTML se solapan unos con otros, por ejemplo, un está en un <p> que está en un que está en el <body>. Si ponemos un escuchador del evento click a todos ellos se ejecutarán todos ellos, pero ¿en qué orden?.

Pues el W3C estableció un modelo en el que primero se disparan los eventos de fuera hacia dentro (primero el <body>) y al llegar al más interno (el <spab>) se vuelven a disparar de nuevo pero de dentro hacia afuera. La primera fase se conoce como fase de captura y la segunda como fase de burbujeo. Cuando ponemos un escuchador con addEventListener el tercer parámetro indica en qué fase debe dispararse:

- **true**: en fase de captura
- **false** (valor por defecto): en fase de burbujeo

Ejemplo

```
target.addEventListener(tipo, listener[, useCapture]);
```