1 Eventos

Imagina que estás escuchando un programa de radio y anuncian: "¡Gran evento! Los alienígenas aterrizaron en la Tierra". Algún oyente podría pensar:"No me lo creo"; otros podrían pensar "Vienen en paz"; y algunos pensarían: "¡Todos vamos a morir!". Del mismo modo, el navegador transmite eventos, y nuestro código tendrá que decidir si sintonizar y escucha los eventos a medida que ocurren o no.

Algunos eventos de ejemplo son los siguientes:

- El usuario hace click en un botón.
- El usuario escribe un carácter en un campo de formulario.
- La página termina de cargarse.

Aquí tenemos la lista de todos los eventos posibles soportados por los objetos HTML y que podremos manejar desde nuestro código de JavaScript. Vemos que la denominación de estos eventos es "igual" a la de HTML pero sin el "on". Algunos eventos de los más utilizados son los siguientes:

Evento	Descripción	Elementos para los que está definido
blur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
change	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
click	Pinchar y soltar el ratón	Todos los elementos
dblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
focus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
keydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
keypress	Pulsar una tecla	Elementos de formulario y <body>
keyup	Soltar una tecla pulsada	Elementos de formulario y <body>
load	La página se ha cargado completamente	<body>
mousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
mousemove	Mover el ratón	Todos los elementos
mouseout	El ratón «sale» del elemento (pasa por encima de otro elemento)	Todos los elementos
mouseover	El ratón «entra» en el elemento (pasa por encima del elemento)	Todos los elementos
mouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
reset	Inicializar el formulario (borrar todos sus datos)	<form>
resize	Se ha modificado el tamaño de la ventana del navegador	<body>
select	Seleccionar un texto	<input>, <textarea>
submit	Enviar el formulario	<form>
unload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

La idea va a ser adjuntar una función de JavaScript llamada "detector de eventos" o "controlador de eventos" a un evento específico y el navegador invocará nuestra función tan pronto como ocurra el evento. Veamos cómo se hace esto.

1.1 Sumario

1.2 Atributos HTML en línea

Agregar atributos específicos a una etiqueta es la forma más perezosa pero menos mantenible; veamos la siguiente línea de código como ejemplo:

```
<div onclick = "alert ('¡Hola!')"> haga clic </div>
```

En este caso, cuando el usuario hace *click* en **div**, se activa el evento *click* y la cadena de código JavaScript contenida en el atributo *onclick* se ejecuta. No hay una función explícita que escuche el evento *click*; sin embargo, "por detrás", se crea una función que contiene el código especificado como un valor del atributo *onclick*.

1.3 Propiedades de elementos

Un buen modo de ejecutar código JavaScript cuando hacemos *click* en un elemento es **vincular una función a la propiedad *onclick* del elemento**. Veamos un ejemplo:

```
<div id="midiv">click</div>

  <script>
    const miElemento = document.getElementById('midiv');
    miElemento.onclick = function () {
      alert('Susto!');
      alert('Y doble susto!');
    };
  </script>
```

Esta es una buena forma de hacerlo porque ayuda a mantener la etiqueta **div** limpia de cualquier código JavaScript.

Hay que tener siempre en cuenta que el HTML vale para el contenido, JavaScript para el comportamiento y CSS para el formato, y se deben mantener siempre estos tres elementos separados lo máximo posible.

Este método tiene el inconveniente de que solo se puede añadir una función al evento, como si un programa de radio tuviese un único oyente. Es cierto que puede pasar mucho dentro de una única función, pero eso no es siempre conveniente, como si todos los oyentes estuviesen en la misma sala...

1.4 Escuchadores de eventos DOM

La mejor forma de trabajar con los eventos del navegador es emplear el enfoque del oyente de eventos descrito en DOM Nivel 2, donde se puede tener muchas funciones escuchando un evento. Cuando un evento se dispara, todas las funciones son ejecutadas. Los oyentes no necesitan saber unos de los otros y pueden funcionar de forma independiente. Pueden enlazarse y salir en cualquier momento, sin afectar a los demás oyentes.

Si tenemos el siguiente código:

```
<p id="closer">final</p>
```

Nuestro código JavaScript puede agregar escuchadores al evento *click* utilizando el método **addEventListener()**.

Agreguemos dos escuchadores del siguiente modo:

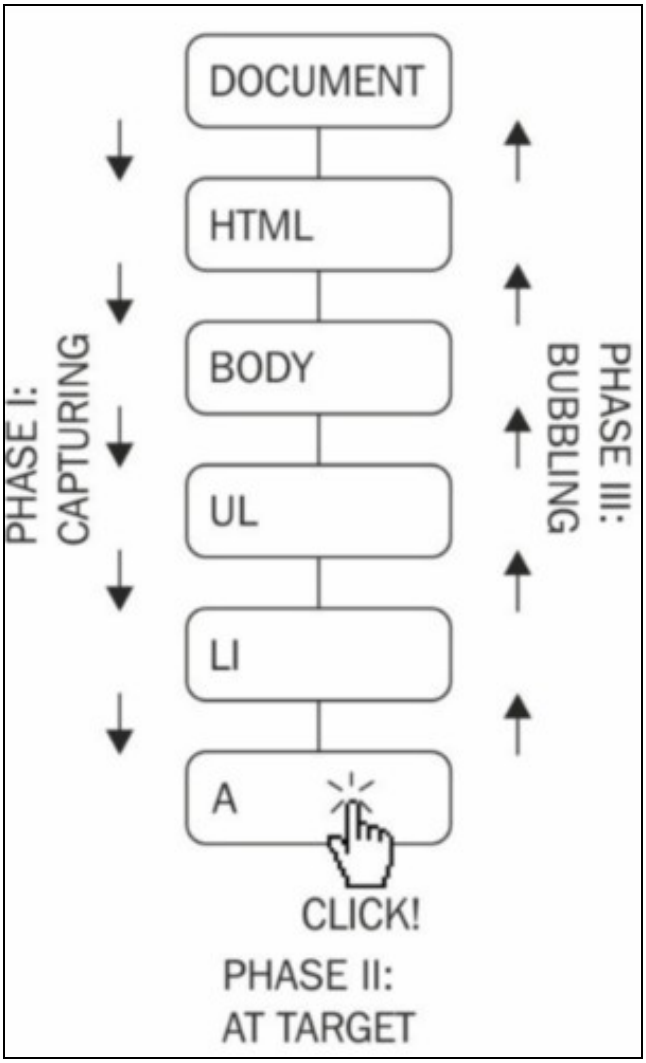
```
<script>
  const miP = document.getElementById("closer");
  miP.addEventListener(
    "click",
    function () {
      alert("Hola");
    }, false
  );

  miP.addEventListener(
    "mouseover",
    function () {
      console.log("Hola consola");
    }, false
  );
```

```
</script>
```

1.5 Capturando y Burbujeando

Cuando llamamos al evento **addEventListener()** pasamos tres parámetros: el evento, la función y un tercer parámetro que siempre ponemos como **false**. Veamos qué quiere decir esto.



Capturando y Burbujeando
En el siguiente código tenemos un enlace en el interior de una lista desordenada:

```
<body>

  <ul>
    <li><a href="https://www.iessanclemente.net">Mi blog</a></li>
  </ul>

</body>
```

Cuando se hace *click* en el enlace, en realidad también se hace *click* en el elemento de la lista, **li**, la lista **ul**, el **body** y el **document** como un todo. Esto se llama 'propagación de eventos'. Un *click* en un enlace también puede verse como un *click* en el **document**. El proceso de propagación de un evento puede ser implementado de las dos formas siguientes:

- **Captura de eventos:** este *click* ocurre primero en el **document**, luego se propaga hacia abajo, al **body**, a la lista, al elemento de la lista y, finalmente, al enlace.
- **Evento burbujeante:** este *click* ocurre en el enlace y luego va subiendo hacia el **document**.

La especificación de eventos DOM de nivel 2 sugiere que los eventos se propagan en tres fases, a saber, captura de eventos, en el objeto y burbujeo. Esto significa que el evento se propaga del **document** al enlace (destino) y luego burbujea de nuevo al **document**. Los eventos de los objetos tienen

una propiedad denominada **eventPhase**, que refleja la fase actual.

Históricamente, **IE** y **Netscape** (trabajando solos y sin un estándar a seguir) implementaron exactamente el método opuesto. **IE** implementó solo burbujeo y **Netscape** solo el método de captura de eventos. Hoy, mucho después de la especificación DOM, los navegadores modernos implementan las tres fases.

Las implicaciones prácticas relacionadas con la propagación de eventos son las siguientes:

- El tercer parámetro para **addEventListener()** especifica si el método de captura debe ser o no usado. Para que su código sea más portátil en todos los navegadores, es mejor configurar siempre el parámetro a **false** y use solo burbujeo.
- Se puede detener la propagación del evento en sus oyentes para que deje de burbujear y nunca llege a **document**. Para hacer esto, puede llamar al método **stopPropagation()** del objeto de evento; Hay un ejemplo más adelante.
- También se puede usar la delegación de eventos. Si tienes diez botones dentro de un **div**, siempre puedes agregar diez oyentes de eventos, uno para cada botón. Sin embargo, una cosa más inteligente es adjuntar solo un oyente para el elemento **div** y una vez que ocurre el evento, verificar qué botón era el objetivo del *click*.

1.6 Parar la propagación

Veamos un ejemplo de cómo se puede parar un evento que está 'burbujeando'. Volvamos al documento de test:

```
<p id="closer">Final</p>
```

Ahora definamos una función que maneja los *clicks* del párrafo del siguiente modo:

```
function paraHandler() {
  alert('¡Click en el párrafo!');
}
```

Ahora, configuremos esta función como un escuchador del evento *click*:

```
const para = document.getElementById('closer');
para.addEventListener('click', paraHandler, false);
```

Ahora, configuremos escuchadores al evento *click* del elemento **body**, el **document** y la ventana del navegador:

```
document.body.addEventListener('click', function () {
  alert('¡Click en el Body!');
}, false);

document.addEventListener('click', function () {
  alert('¡Click en el doc!');
}, false);

window.addEventListener('click', function () {
  alert('¡Click en el window!');
}, false);
```

Ahora, si se hace *click* en el párrafo, deberías ver cuatro alertas, y en este orden: **Párrafo ? body ? document ? window**

Esto ilustra cómo el mismo evento *click* se propaga (burbujeo).

El método contrario de **addEventLister()** es **removeEventListener()**, y éste acepta, exactamente, los mismos parámetros.

Ahora, borremos el escuchador de eventos del párrafo escribiendo el siguiente código en la consola:

```
> para.removeEventListener('click', paraHandler, false);
```

Si ahora hacemos *click*, sólo veremos alertas para **body ? document ? window** pero no la del párrafo.

Ahora, paremos la propagación del evento. En la función que se añadió para recibir el objeto evento como parámetro se puede llamar al método **stopPropagation()** del siguiente modo:

```
function paraHandler(e) {
  alert('¡Realizado un click en el párrafo!');
```

```
    e.stopPropagation();
}
```

Ahora, cuando se haga *click* en el párrafo, sólo se verá una alerta porque se para el burbujeo...

1.7 Evitar el comportamiento por defecto

Algunos eventos del navegador tienen un comportamiento predefinido. Por ejemplo, hacer *click* en un enlace hace que el navegador cargue otra página. Se pueden modificar los escuchadores en un enlace en concreto y también se puede desactivar el comportamiento predeterminado llamando al método **preventDefault()** en el objeto del evento.

Vamos a hacer un ejemplo donde preguntemos a los visitantes de la página "¿Estás seguro de que quieres seguir este enlace?" cada vez que hacen *click* en un enlace. Si el usuario hace *click* en Cancelar se llama al método **preventDefault()**:

```
//Todos los enlaces
const enlaces = document.getElementsByTagName('a');
for (let i = 0; i < enlaces.length; i++) { //recorremos todos
  enlaces[i].addEventListener(
    'click', //evento click
    function (e) { //manejador
      if (!confirm('¿Quieres ir a este enlace?')) {
        e.preventDefault();
      }
    },
    false //No utilizar captura de eventos
  );
}
```

1.8 Objeto evento

El objeto evento contiene toda la información relevante del evento en sí. Por ejemplo, los eventos **click** tiene las propiedades **clientX** y **clientY**, que nos dan las coordenadas donde el *click* se produce.

```
//Agregamos escuchador al 'document' de, por ejemplo, un HTML vacío
document.addEventListener('click', coordenadas, false);

function coordenadas(e) {

    //Recojemos las coordenadas x e y donde hacemos click
    const x = e.clientX;
    const y = e.clientY;

    console.log(x + ' , ' + y);
}
```

Otra propiedad del evento *click* es **target** que recoge el elemento donde se realizó el *click*.

1.9 Tipos de eventos

Ahora ya se sabe cómo manejar eventos en los navegadores. Sin embargo, en todos los ejemplos anteriores solo se utiliza el evento *click*. ¿Qué otros eventos están sucediendo por ahí?

Como probablemente se pueda adivinar, diferentes navegadores proporcionan diferentes eventos. Hay un subconjunto de eventos comunes en todos los navegadores y otros específicos del navegador. Para obtener una lista completa de eventos, debe consultar la documentación del navegador, pero aquí hay una selección de eventos que se dan en todos los navegadores:

- **Eventos de ratón.**

  - **mouseup**, **mousedown**, **click** (la secuencia es **mousedown-up-click**), **dblclick**
  - **mouseover**, **mouseout**, **mousemove**

- **Keyboard events.**

  - **keydown**, **keypress**, **keyup** (ocurre en esa secuencia)

- **Loading/window events.**

  - **load** (una imagen o una página y todos sus componentes ya están cargados), **unload** (el usuario deja la página), **beforeunload** (el *script* puede proporcionar al usuario una opción para detener la descarga).
  - **abort**, **error** (un error de JavaScript).
  - **resize** (la ventana del navegador cambia de tamaño), **scroll**, **contextmenu**

- **Form events.**

  - **focus** (entrar en un elemento de un formulario), **blur** (salir de un elemento de un formulario).
  - **change** (dejar un campo después de que su valor cambiara), **select** (seleccionar texto en un campo *text*).
  - **reset** (eliminar todas las entradas del usuario), **submit** (enviar el formulario).

Además, los navegadores modernos proporcionan eventos de arrastre (**dragstart**, **dragend**, **drop**...entre otros) y los dispositivos táctiles proporcionan **touchstart**, **touchmove** y **touchend**.

Tabla resumen de eventos

Atributo	El evento se produce cuando...	IE	F	O	W3C
onblur	Un elemento pierde el foco.	3	1	9	Si
onchange	El contenido de un campo cambia.	3	1	9	Si
onclick	Se hace click con el ratón en un objeto.	3	1	9	Si
ondblclick	Se hace doble click con el ratón sobre un objeto.	4	1	9	Si
onerror	Ocurre algún error cargando un documento o una imagen.	4	1	9	Si
onfocus	Un elemento tiene el foco.	3	1	9	Si
onkeydown	Se presiona una tecla del teclado.	3	1	No	Si
onkeypress	Se presiona una tecla o se mantiene presionada.	3	1	9	Si
onkeyup	Cuando soltamos una tecla.	3	1	9	Si
onload	Una página o imagen terminaron de cargarse.	3	1	9	Si
onmousedown	Se presiona un botón del ratón.	4	1	9	Si
onmousemove	Se mueve el ratón.	3	1	9	Si
onmouseout	Movemos el ratón fuera de un elemento.	4	1	9	Si
onmouseover	El ratón se mueve sobre un elemento.	3	1	9	Si
onmouseup	Se libera un botón del ratón.	4	1	9	Si
onresize	Se redimensiona una ventana o frame.	4	1	9	Si
onselect	Se selecciona un texto.	3	1	9	Si
onunload	El usuario abandona una página.	3	1	9	Si

Volver