Eventos. Introducción

Eventos son acciones que se producen en nuestra aplicación: ya sean generadas directamente por el usuario o por nuestra aplicación. Los eventos se pueden dividir en categorías:

- **Eventos producidos por el ratón**: click, dblclick, mouseenter, mouseleave, mousedown, mouseup, mousemove, drag, dragstart, dragend, dragenter, dragleave, dragover, drop, …
- **Eventos producidos por el teclado**: keydown/keypress, keyup.
- **Eventos producidos en los elementos de un formulario**: focus, blur, submit, reset, change, input, select, …
- **Eventos relacionados con la página**: load, unload, resize, …
- **Eventos de bases de datos**: abort, blocked, complete, error, success, upgradeneeded, versionchange, …
- **Eventos de peticiones asíncronas**: readystatechange, …
- **Eventos de conexión de red**: disabled, enabled, offline, online, statuschange
- **Eventos de cambios del DOM**.
- **Eventos de medios**: canplay, durationchange, pause, play, ..

Eventos. Introducción

- **Eventos de animaciones**: animationend, animationiteration, animationstart..
- **Eventos de impression:** afterprint, beforeprint, …
- **Eventos de clipboard:** copy, paste, cut, select, …

JavaScript permite programar que se ejecute una tarea cuando se produzca un evento. Es lo que se conoce en programación como **capturar un evento**. Es decir, se indica que cuando se produzca un evento se ejecute una función.

Acción por defecto de la acción

Un evento puede generar la ejecución de una función si así se ha programado (captura del evento). Y además, ese mismo evento puede generar la ejecución de una tarea propia de ese evento, es a lo que se llama acción por defecto.

Por ejemplo cuando en un enlace se hace click se carga la página a la que apunta dicho enlace. Esto es una acción por defecto del click en un enlace. Además, si está programado que cuando se haga click en el enlace se ejecute una función (por la captura del evento click), ocurrirán las dos funciones así:

1)Se ejecutará la función asociada al evento.
2)La acción por defecto se ejecutará después (*cargarse la página a la que apunta el enlace*)

2.1. Modelo de registro de eventos en línea

La captura del evento puede ser incluido como un atributo más a la etiqueta HTML, como en este ejemplo:

```
<a id=”enlace” href="pagina.html" onclick="alertar()">Pulsa aqui</a>

function alertar() {

        alert("Has pulsado en el enlace");

}
```

No se recomienda utilizar el modelo de registro de eventos en línea. Ya que lo que se recomienda es separar completamente la programación JavaScript de la estructura HTML.

Modo de funcionamiento de los eventos

El orden de ejecución es el siguiente:

1.El script se ejecutará primero (función `alertar()`)

2.La acción por defecto se ejecutará después (*cargarse la página* `pagina.html`)

Ejercicio 10

2.1. Modelo de registro de eventos en línea

Evitar la acción por defecto desde el código HTML

Desde el código HTML se puede evitar la ejecución de la acción por defecto. En este caso sería la carga de la página HTML.

```
<a id=”enlace” href="pagina.html" onclick="alertar(); return
false">Pulsa aqui</a>
```

Con el return false estamos diciendo al navegador que no realice la acción del objeto `<a>`, que es cargar la página `pagina.html`

Ejercicio 11

Evitar la acción por defecto desde el código JavaScript

```
<a id=”enlace” href="pagina.html" onclick="return
preguntar()">Pulsa aqui</a>

function preguntar() {
     return confirm("¿Desea realmente ir a esa dirección?");
}
```

Ejercicio 12 y 13

2.2. Modelo de registro de eventos tradicional

El modelo de registro de eventos tradicional se **basa en la separación del código JavaScript de la estructura HTML.**

El evento pasa a ser una propiedad del elemento, y puede ser manejado por código JavaScript.

No está admitido por W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día.

Una forma de hacerlo sería así:

```
<a id=”enlace” href="pagina.html">Pulsa aqui</a>

<script>
  enlace.onclick=funcion;
  function funcion () {
     alert(“Vas a acceder a una pagina web”);
  }
</script>
```

Nota importante → si se pusiera: `enlace.onclick=funcion();`

Se ejecutaría `funcion()` y el resultado se asignaría al atributo `onclick` de enlace

Ejercicio 17

Para eliminar un registro de evento de un elemento u objeto: `elemento.onclick=null;`

En nuestro ejemplo sería: `enlace.onclick=null;`

2.2. Modelo de registro de eventos tradicional

Además de la ventaja de separación de código, nos permite lanzar el evento de forma manual. Por ejemplo:

```
elemento.onclick();
```

Utilización del objeto this en modelo de registro de eventos tradicional:

```
<a id=”enlace” href="pagina.html">Pulsa aqui</a>

<script>
  enlace.onclick=funcion;

  function funcion () {
     alert(“Vas a acceder a la pagina web” + this.href);
  }
//this es el elemento que tiene programado el evento

</script>
```

2.3. Modelo de registro avanzado de eventos W3C

Dado que W3C no estandarizó el modelo tradicional ofreció como alternativa el modelo de registro avanzado de eventos, con el método: `addEventListener()`

Este método tiene **tres argumentos: el tipo de evento, la función a ejecutar y un valor booleano (true o false).** Este valor booleano se utiliza para indicar cuando se debe capturar el evento: en la fase de captura (true) o de burbujeo (false) (se verá más adelante la diferencia entre las dos fases)

```
elemento.addEventListener('evento', función, false|true)
```

Por ejemplo:

¡¡¡Observa!!! NO SE PONE on

```
enlace.addEventListener('click', alertar, false)
function alertar() {
     alert("Te conectaremos con la página que es un poco pesada:
"+this.href);
}  //this es el objeto enlace que es quien tiene programado el evento
```

Podemos añadir varias funciones para el mismo evento:

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
enlace.addEventListener('click',alertar,false);

enlace.addEventListener('click',avisar,false);

enlace.addEventListener('click',chequear,false);
```

Por lo tanto, cuando hagamos clic en “enlace” se disparará la llamada a las tres funciones.

2.3. Modelo de registro avanzado de eventos W3C

¿Eliminar un evento de un elemento?

Para eliminar un evento asociado a un objeto HTML se puede utilizar el método :

```
elemento.removeEventListener('evento', función, false|true);
```

Ejemplo:

```
enlace.removeEventListener(“click”, alertar, false);
```

Ejercicio 23

2.3. Modelo de registro avanzado de eventos W3C

También se pueden usar funciones anónimas (sin nombre de función), con el modelo W3C:

```
element.addEventListener('click', function () {
     this.style.backgroundColor = '#cc0000';
}, false);
```

CUIDADO: No se puede utilizar la función flecha.

Función flecha en registro de eventos

- **IMPORTANTE: No se debe utilizar en la función manejadora de un evento: `this` será `window` en lugar del objeto que tiene definida la función manejadora.**

```
b.addEventListener("click",function (){console.log(this)})
// cuando se hace click en el elemento b sale el elemento b en consola.


b.addEventListener("click", ()=>{console.log(this)})
// cuando se hace click en el elemento b sale Window en consola.
```

2.3. Modelo de registro avanzado de eventos W3C

Para anular la ejecución del evento por defecto de un elemento

Para evitar que se ejecute el evento por defecto de un elemento según el modelo de registro avanzado de eventos se utiliza el método **`event.preventDefault();`**

`event` es un objeto que guarda toda la información del evento que se ha producido en el elemento.
`preventDefault` es un método del objeto **`event.`**

Hay eventos que no permiten ejecutar `preventDefault()` , por ejemplo `load`. Los eventos tienen una propiedad de solo lectura llamada **`cancelable`** que indica si la acción por defecto se puede cancelar o no.

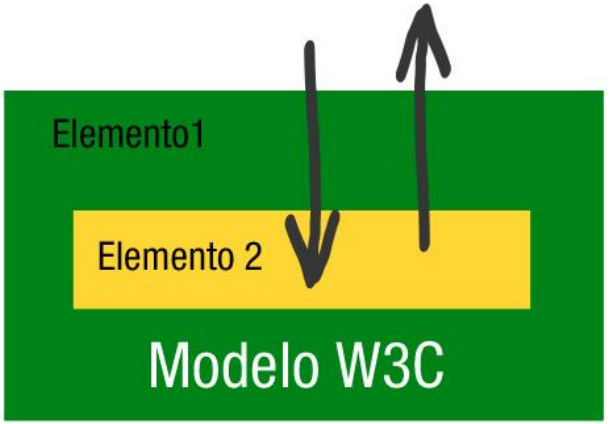
En este ejemplo se hace así (en el caso de los enlaces el evento por defecto es abrir la página indicada en href):

```
<a id="enlace" href="https://w3schools.com/">Enlace</a>

enlace.addEventListener("click", anularAccionDefecto);

function anularAccionDefecto(event){

    event.preventDefault();

}
```

Ejercicios 24 y 25

2.4. Orden de disparo de los eventos

Si tenemos dos elementos, uno contenido dentro de otro, y tenemos programado el mismo tipo de evento para ambos. Al producirse el evento en el de dentro, se va a producir también un evento en el de fuera, ¿cuál de los eventos se disparará primero?



Según el modelo W3C

Si se produce un evento en Elemento 2 también se está produciendo en Elemento 1. El modelo W3C divide el proceso de disparo en dos fases:

- **La fase captura (de fuera a dentro):** que empieza en Elemento 1 y termina en Elemento 2.
- **La fase de burbujeo (de dentro a fuera):** que empieza en Elemento 2 y termina en Elemento 1.

Tú podrás decidir cuando quieres que se ejecute la función manejadora del evento: en la fase de captura o en la fase de burbujeo. **El tercer parámetro de addEventListener te permitirá indicar si lo haces en la fase de captura (`true`), o en la fase de burbujeo(`false`).**

2.4. Orden de disparo de los eventos

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,false);

elemento2.addEventListener('click',hacerAlgo2,false);
```

Si el usuario hace clic en el elemento2 se producirá el evento click de elemento2 y de elemento 1. El elemento destino es elemento2, que es en el que se ha originado el evento (el más interno).

El orden de disparo será el siguiente:

- El evento de clic comenzará en la fase de captura (de fuera al elemento destino)
  - El evento comprueba si hay algún ancestro del elemento2 que tenga el evento capturado para la fase de captura (true). El evento encuentra un elemento1 que tiene capturado el evento click para que se ejecute hacerAlgo1() en la fase de burbujeo por lo que no hace nada.
  - El evento viajará hacia el destino (elemento2) que tiene capturado el evento click para que se ejecute hacerAlgo2() en la fase de burbujeo, por lo que no hace nada.
- Entonces el evento pasa a la fase de burbujeo (del elemento destino hacia fuera)
  - El evento busca si está capturado en elemento2 el evento en fase de burbujeo y ve que sí, por lo que ejecuta su manejador que es hacerAlgo2().
  - El evento viaja hacia arriba de nuevo y chequea si algún ancestro (Elemento1) tiene programado este evento para la fase de burbujeo, ve que es así y ejecuta su manejador que es hacerAlgo1().

2.4. Orden de disparo de los eventos

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);

elemento2.addEventListener('click',hacerAlgo2,true);
```

Si el usuario hace clic en el elemento2 se producirá el evento click de elemento2 y de elemento 1. El elemento destino es elemento2, que es en el que se ha originado el evento (el más interno)

El orden de disparo será el siguiente:

- El evento de clic comenzará en la fase de captura (de fuera al elemento destino)
  - El evento comprueba si hay algún ancestro del elemento2 que tenga el evento capturado para la fase de captura (true). El evento encuentra un elemento1 que tiene capturado el evento click para que se ejecute hacerAlgo1() en la fase de captura por lo que ejecuta hacerAlgo1()
  - El evento viajará hacia el destino (elemento2) que tiene capturado el evento click para que se ejecute hacerAlgo2() en la fase de captura, por lo que se ejecuta hacerAlgo2()
- Entonces el evento pasa a la fase de burbujeo (del elemento destino hacia fuera)
  - El evento busca si está capturado en elemento2 el evento en fase de burbujeo y ve que no, por lo que no hace nada
  - El evento viaja hacia arriba de nuevo y chequea si algún ancestro (Elemento1) tiene programado este evento para la fase de burbujeo, ve que no por lo que no hace nada.

Haz este ejercicio y saca conclusiones.

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_element_addeventlistener_capture