LENGUAJES DE MARCAS

EVENTOS en JavaScript

Pablo Matías Garramone Ramírez

EVENTOS

- La programación tradicional, suele ser secuencial, es decir, se van ejecutando las líneas de código una detrás de la otra.
- Sin embargo, cada vez más, aparecen lenguajes basados en eventos. Es decir, una interacción entre la aplicación y el usuario, que ante acciones del usuario se lanzan diferentes acciones en la aplicación.
- JavaScript tiene ambas modalidades, secuencial y por eventos.
- También existen eventos que no dependen del usuario, como por ejemplo cargar la página o salir de ella que también son eventos sobre los que JavaScript puede actuar.
- Aquí tienes un listados de todos los eventos, a continuación veremos los más importantes.
- https://www.w3schools.com/jsref/dom_obj_event.asp

Pablo Matías Garramone Ramírez

Eventos de la página (body)

- Estos eventos son producidos en el documento HTML. Normalmente afectan al elemento **body.**
- **load** → Este evento se lanza cuando el documento HTML ha terminado de cargarse. Es util para realizar acciones que requieran que el DOM haya sido completamente cargado (como consultar o modifcar el DOM).
- **unload** → Ocurre cuando el documento es destruido, por ejemplo, después de cerrar la pestaña donde la pagina estaba cargada.
- **beforeunload** → Ocurre justo antes de cerrar la pagina. Por defecto, un mensaje pregunta al usuario si quiere realmente salir de la pagina, pero hay otras acciones que pueden ser ejecutadas.
- **resize** → Este evento se lanza cuando el tamaño del documento cambia (normalmente se usa si la ventana se redimensiona)

Pablo Matías Garramone Ramírez

Eventos de teclado

- **keydown** → El usuario presiona una tecla. Si la tecla se mantiene pulsada durante un tiempo, este evento se generara de forma repetida.
- **keyup** → Se lanza cuando el usuario deja de presionar la tecla
- **keypress** → Mas o menos lo mismo que **keydown**. Acción de pulsar y levantar.

Pablo Matías Garramone Ramírez

Eventos del ratón

- **click** → Este evento ocurre cuando el usuario pulsa un elemento (presiona y levanta el dedo del boton → mousedown + mouseup). También normalmente se lanza cuando un evento táctil de toque (tap) es recibido.
- **dblclick** → Se lanza cuando se hace un doble click sobre el elemento
- **mousedown** → Este evento ocurre cuando el usuario presiona un botón del ratón
- **mouseup** → Este evento ocurre cuando el usuario levanta el dedo del botón del ratón
- **mouseenter** → Se lanza cuando el puntero del ratón entra en un elemento
- **mouseleave** → Se lanza cuando el puntero del ratón sale de un elemento
- **mousemove** → Este evento se llama repetidamente cuando el puntero de un ratón se mueve mientras esta dentro de un elemento

Pablo Matías Garramone Ramírez

Eventos Touch

- **touchstart** → Se lanza cuando se detecta un toque en la pantalla táctil
- **touchend** → Se lanza cuando se deja de pulsar la pantalla táctil
- **touchmove** → Se lanza cuando un dedo es desplazado a través de la pantalla
- **touchcancel** → Este evento ocurre cuando se interrumpe un evento táctil.

Pablo Matías Garramone Ramírez

Eventos de Formulario

- **focus** → Este evento se ejecuta cuando un elemento (no solo un elemento de un formulario) tiene el foco (es seleccionado o esta activo).
- **blur** → Se ejecuta cuando un elemento pierde el foco.
- **change** → Se ejecuta cuando el contenido, selección o estado del checkbox de un elemento cambia (solo <input>, <select>, y <textarea>)
- **input** → Este evento se produce cuando el valor de un elemento <input> o <textarea> cambia.
- **select** → Este evento se lanza cuando el usuario selecciona un texto de un <input> o <textarea>.
- **submit** → Se ejecuta cuando un formulario es enviado (el envío puede ser cancelado).

Pablo Matías Garramone Ramírez

Sintaxis	Descripción	Evento de...
onAbort	Ocurre cuando el usuario aborta la carga de una imagen (pulsando sobre un enlace o sobre el botón **Stop**).	*Image*.
onBlur	Ocurre cuando el usuario quita el foco de un elemento de un formulario, de una ventana o de un *frame*.	*Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, Textarea* y *Window*.
onChange	Ocurre cuando un select, text o textArea pierde el foco y su valor ha sido modificado.	*FileUpload, Select, Text* y *Textarea*.
onClick	Ocurre cuando el usuario hace un clic sobre un enlace o un elemento de un formulario.	*Button, Checkbox, Link, Radio, Reset* y *Submit*.
onDblClick	Ocurre cuando el usuario hace un doble clic sobre un enlace o un elemento de un formulario.	*Document, Area* y *Link*.
onFocus	Ocurre cuando se produce un error en la carga de un documento o una imagen.	*Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea* y *Window*.
onKeyDown	Ocurre cuando el usuario pulsa una tecla.	*Document, Image, Link* y *Textarea*.
onKeyPress	Ocurre cuando el usuario pulsa o mantiene pulsada una tecla.	*Document, Image, Link* y *Textarea*.
onKeyUp	Ocurre cuando el usuario libera una tecla.	*Document, Image, Link* y *Textarea*.
onLoad	Ocurre cuando el navegador ha terminado de cargar la página.	*Image, Layer* y *Window*.
onMouseDown	Ocurre cuando el usuario pulsa un boton del ratón.	*Button, Document* y *Link*.
onMouseMove	Ocurre cuando el usuario mueve el ratón.	Se debe asociar explícitamente a algún objeto.
onMouseOut	Ocurre cuando el usuario mueve el ratón fuera de un objeto.	*Area, Layer* y *Link*.
onMouseOver	Ocurre cuando el usuario mueve el ratón dentro de un objeto.	*Area, Layer* y *Link*.
onMove	Ocurre cuando se mueve una ventana.	*Window*.
onReset	Ocurre cuando el usuario pulsa el botón *reset* de un formulario.	*Form*.
onResize	Ocurre cuando el usuario cambia el tamaño de la ventana.	*Window*.
onSelect	Ocurre cuando el usuario selecciona algún texto en un objeto *text* o *textarea*.	*Text* y *Textarea*.
onSubmit	Ocurre cuando el usuario envía un formulario.	*Form*.

Eventos Más utilizados

- 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.
- Es interesante resaltar, que evento onload es uno de los más utilizados ya que, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.
- Por lo que una técnica de las 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 e incluir ese código dentro de esta estructura:

```
window.onload = function() {
... // INCLUIR AQUÍ LA ASIGNACIÓN A LOS MANEJADORES
    // Ejemplo: document.getElementById("pinchable").onclick = muestraMensaje;
}
```

Pablo Matías Garramone Ramírez

Manejadores de Eventos

- Los eventos los tenemos que asociar a una función o a código JavaScript para que pueda responder al mismo.
- Estas funciones o código asociado se denomina “Manejador de eventos” (Event handler).
- Existes varias formas de usar manejadores:
  - La forma clásica, que podemos usarla en:
    - Código JavaScript dentro de un atributo en el propio elemento HTML (poco profesional)
    - Definiendo el evento dentro del propio elemento HTML pero el manejador en una función externa
    - Manejadores Semánticos asignados mediante DOM sin tocar nada del HTML (más profesional)
  - Usando eventos listener (Más recomendado), que permiten múltiples respuestas manejadoras a un mismo evento.

Pablo Matías Garramone Ramírez

Forma Clásica I. Manejadores con atributos XHTML

- Es la forma menos profesional. Y solo se usa para casos muy sencillos.
- Se basa en añadir un atributo en el elemento XHTML que queramos que tenga el evento y añadirle el prefijo “on” al nombre del evento a capturar. Por ejemplo, del evento click, sería onClick.
- Puedes poner el código javaScript que quieras.

```
<!DOCTYPE>
  <html>
        <head>
              <title>Ejemplo JS</title>
        </head>
        <body>
              <div id="div1” onmouseover="console.log('Acabas de pasar el ratón por
                  encima'); if (2>1) alert (‘hola’);">
              <p>
                  <input type="text" onclick="alert('!Me has pulsado!'); alert(‘adios’);" />
              </p>
              </div>
        </body>
  </html>
```

Pablo Matías Garramone Ramírez

Variable THIS

- **this** en JavaScript hace referencia al elemento html que provocó el evento.
- Ejemplo: Si tenemos este DIV, donde queremos que cuando se pase el ratón por encima el color sea NEGRO y cuando salga del DIV vuelva a ser GRIS.

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

- El código queda muy largo y engorroso, se puede abreviar de este modo:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
   onmouseover="this.style.borderColor='black';"
   onmouseout="this.style.borderColor='silver';">
        Sección de contenidos...
</div>
```

Pablo Matías Garramone Ramírez

Forma Clásica II. Eventos con funciones externas

- Cuando el código es más complejo es conveniente usar funciones.
- El inconveniente que se sigue invocando desde el XHTML.

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

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

- Otro problema es que desde la función no tenemos acceso a la variable this (El elemento XHTML), tenemos que enviarsela. También le podemos pasar la variable event (Propiedades que dan información sobre el evento, que veremos más adelante)
- Archivo: ejemplo1.html

```
<input type="text" id="input1" onclick="inputClick(this, event)" />
```

- Archivo: ejemplo1.js

```
function inputClick(element, event) {
// Mostrara "Un evento click ha sido detectado en #input1"
alert("Un evento" + event.type + " ha sido detectado en #" +
    element.id);
}
```

Pablo Matías Garramone Ramírez

Forma Clásica III. Eventos semánticos (usando el DOM)

- En este caso, tal como hacemos con el CSS y no tocar el HTML, de manera que todo se hace en un fichero externo o fuera del HTML, dejando este código limpio y legible.
- Podemos añadir un manejador (función) de evento desde código, accediendo a la propiedad correspondiente (onclick, onfocus, …).
- También podemos asignarles un valor nulo si queremos dejar de escuchar algún evento.
- Deberemos:
  1. Asignar un identificador único al elemento XHTML mediante el atributo id.
  2. Crear una función de JavaScript encargada de manejar el evento.
  3. Asignar la función a un evento concreto del elemento XHTML mediante DOM.
- El código resultante es más limpio, no tocamos el HTML y además podemos usar la variable this.
- Ejemplo (en este caso declaro la función en la misma asignación del evento):

```
let input = document.getElementById("input1");
input.onclick = function(event) {
        alert("Un evento " + event.type + " ha sido detectado en " + this.id);
}
```

Pablo Matías Garramone Ramírez

Forma Recomendada. Events listener

- Es parecido a este último caso de los Eventos semánticos usando el DOM, pero tiene una ventaja sobre el resto. Pueden gestionar varias funciones manejadoras para un mismo evento.
- Para añadir un event listener debemos usar el método addEventListener sobre el elemento que deseamos gestionar un evento. Tendremos 3 parámetros:
  - El nombre del evento (Sin el prefijo ON)
  - La función del evento
  - Si el manejador se emplea en fase de capture (Será TRUE) o en fase de bubbling (Será FALSE) (Lo detallamos más adelante).

```
let input = document.getElementById("input1");
input.addEventListener('click', function(event) {
    alert("Un evento " + event.type + " ha sido detectado
    en " + this.id);
});
```

Pablo Matías Garramone Ramírez

Forma Recomendada. Events listener II

- Como decía la ventaja que podemos asociar varias funciones al mismo evento.

```
let inputClick = function(event) {
console.log("Un evento " + event.type + " ha sido detectado en " +
    this.id);
};
let inputClick2 = function(event) {
console.log("Yo soy otro manejador para el evento click!");
};
let input = document.getElementById("input1");
// Anadimos ambos manejadores. Al hacer clic, se ejecutarían
    ambos por orden.
input.addEventListener('click', inputClick);
input.addEventListener('click', inputClick2);
```

- Podemos desasociar el manejador de eventos con removeEventListener.

```
// Así es como se elimina el manejador de un evento
input.removeEventListener('click', inputClick);
input.removeEventListener('click', inputClick2);
```

Pablo Matías Garramone Ramírez

Propagación de Eventos (bubbling y capturing)

- Además de los eventos básicos, existe un mecanismo llamado flujo de evento o propagación de eventos, que se producen cuando hay elementos en una web que se solapan con otros, es decir, están contenidos unos en otros.
- En estos casos, la propagación de eventos permite que varios elementos diferentes respondan al mismo evento.
- Tendremos dos ordenes posibles:
  - Del elemento más específico al menos específico (Evento bubbling)
  - Del elemento menos específico al más específico (Evento capturing)

Pablo Matías Garramone Ramírez

Evento bubbling

- Si hacemos click sobre el elemento rojo , se imprimirá solo “Has pulsado: div1”.
- Sin embargo, si pulsamos sobre el elemento azul (el cual esta dentro del elemento rojo) imprimirá ambos mensajes (#div2 primero).
- En conclusión, por defecto el elemento el cual esta al frente (normalmente un elemento hijo) recibe el evento **primero** y entonces pasa a ejecutar los manejadores que contiene.

Archivo: ejemplo1.html

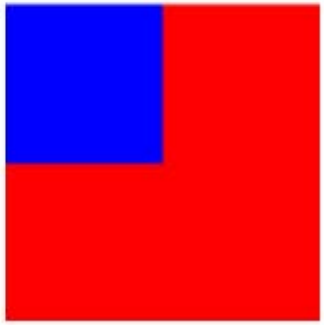
```
<div id="div1" style="background-color: red; width: 200px; height: 200px;">
  <div id="div2" style="background-color: blue; width: 100px; height: 100px;"></div>
</div>
```

Archivo: ejemplo1.js

```
let divClick = function(event) {
   console.log("Has pulsado: " + this.id);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
```



Pablo Matías Garramone Ramírez

Evento capturing

- Ahora con 3 div, cambiamos la función addEventListener y le ponemos TRUE como tercer parámetros, cambiará el orden, e iremos del elemento menos específico al más específico (del padre al hijo) al pulsar sobre el div azul.

Archivo: ejemplo1.html

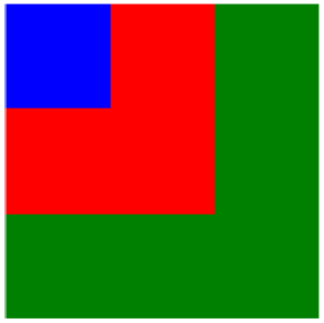
```
<div id="div1" style="background-color: green; width: 150px; height: 150px;">
    <div id="div2" style="background-color: red; width: 100px; height: 100px;">
        <div id="div3" style="background-color: blue; width: 50px; height: 50px;"></div>
    </div>
</div>
```

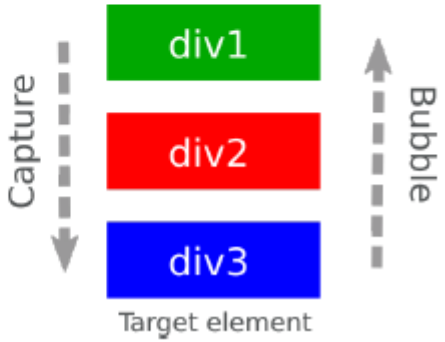
Archivo: ejemplo1.js

```
let divClick = function(event) {
  // eventPhase: 1 -> capture, 2 -> target (objetivo), 3 -> bubble
  console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");
let div3 = document.getElementById("div3");

div1.addEventListener('click', divClick, true );
div2.addEventListener('click', divClick, true );
div3.addEventListener('click', divClick, true );
```





Pablo Matías Garramone Ramírez

- **El resultado sería:**

```
Has pulsado: div1. Fase: 1
Has pulsado: div2. Fase: 1
Has pulsado: div3. Fase: 2
```

Y si quitamos el TRUE:

```
Has pulsado: div3. Fase: 2 → Target element
Has pulsado: div2. Fase: 3 → Bubbling
Has pulsado: div1. Fase: 3 → Bubbling
```

Parar la propagación

- Podemos llamar al método **stopPropagation** en el evento, no continuara la propagación (si es en la fase de captura) o en (la fase de propagación o bubbling).

```
let divClick = function(event) {
   // eventPhase: 1 -> capture, 2 -> target (clicked), 3 -> bubble
   console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
   event.stopPropagation();
};
```

- Ahora, cuando hacemos click el elemento DIV azul imprimirá solo un mensaje:
  - Si el tercer parámetro es FALSE o no lo pones: “Has pulsado: div3. Fase: 2”
  - Si el tercer parámetro es TRU: “Has pulsado: div1. Fase: 1” (el elemento padre previene a los hijos de recibirlo).

Pablo Matías Garramone Ramírez

El objeto Event

- Es creado por JavaScript y pasado por parámetro al manejador automáticamente, lo ponemos como parámetro y no es necesario enviárselo desde el llamador.
- Cuando acaban los manejadores de dicho evento JavaScript destruye el objeto Event correspondiente.
- Este objeto tiene algunas propiedades generales (independientemente del evento específico) y otras particulares del evento (Por ejemplo, un evento de ratón tendrá coordenadas del cursor…).

Objeto Event. Propiedades generales

- **target** → El elemento que lanza el evento (si fue pulsado, etc...).
- **type** → El nombre del evento: 'click', 'keypress', …
- **cancelable** → Devuelve true o false. Si el evento se puede cancelar significa que llamando a **event.preventDefault()** se puede anular la acción por defecto (El envío de un formulario, el click de un link, etc…).
- **bubbles** → Devuelve cierto o falso dependiendo de si el evento se esta propagando.
- **preventDefault()** → Este método previene el comportamiento por defecto (cargar una pagina cuando se pulsa un enlace, el envío de un formulario, etc.)
- **stopPropagation()** → Previene la propagación del evento.
- **stopImmediatePropagation()** → Si el evento tiene mas de un manejador, se llama a este método para prevenir la ejecución del resto de manejadores.

Objeto Event. Del ratón (MouseEvent)

- **button** → Devuelve el botón del ratón que lo ha pulsado (0: botón izquierdo, 1: la rueda del raton, 2: botón derecho).
- **clientX, clientY** → Coordenadas relativas del ratón en la ventana del navegador cuando el evento fue lanzado.
- **pageX, pageY** → Coordenadas relativas del documento HTML, si se ha realizado algún tipo de desplazamiento (scroll), este será añadido (usandclientX y clientY no se añade).
- **screenX, screenY** → Coordenadas absolutas del raton en la pantalla.
- **detail** → Indica cuantas veces el botón del ratón ha sido pulsado (un click, doble, o triple click).

Pablo Matías Garramone Ramírez

Objeto Event. Del teclado (KeyboardEvent)

- **key** → Devuelve el nombre de la tecla pulsada.
- **keyCode** → Devuelve el codigo del caracter Unicode en el evento **keypress**, **keyup** o **keydown**.
- **altKey, ctrlKey, shiftKey, metaKey** → Devuelven si las teclas "**alt**", "**control**", "**shift**" o "**meta**" han sido pulsadas durante el evento (Bastante útil para las combinaciones de teclas como ctrl+c). El objeto **MouseEvent** también tiene estas propiedades.

Pablo Matías Garramone Ramírez