
ICARO
Nuestro compromiso es con el futuro.

Front End I

Clase 2

Eventos

Uno de los mayores desafíos a la hora de introducir el dinamismo en una web es la gestión de los **eventos**, ya que estos son **asíncronos** por naturaleza, por lo que no puedes saber de antemano por ejemplo, cuántos segundos tardará un usuario en clickear un botón...
Lo que sumado a un funcionamiento peculiar con propagaciones, hace que el dominio de los eventos sea una tarea que requiere atención.

¿Cómo funcionan?

Básicamente podemos seleccionar un elemento de HTML, y **suscribir** uno de los posibles **eventos** de los que dispone:

- Eventos de ratón (mouse)
- Eventos de teclado
- Y muchos más...

Una vez hemos definido estos detalles, en dónde *escuchamos* y qué esperamos, solo queda definir qué haremos cuando esto ocurra.
Lógicamente al tratarse de asincronía, debemos definir una función, ya sea en línea o reutilizable para gestionarlo todo.
Si necesitamos información adicional sobre el **evento** sucedido, el propio sistema mandará un **objeto** con detalles clave sobre el evento, como **argumento** a la hora de ejecutar nuestro **callback.**

Eventos

Utilizando eventos

Existen dos formas básicas de añadir eventos a nuestra aplicación. Una es por medio de html con atributos como **onclick**, y la otra desde el JavaScript, haciendo uso de métodos como **.addEventListener().**
Cuando el código se va haciendo más complejo mucho mejor separar JavaScript, HTML y CSS.
Así evitamos el antipatrón de mezclar JavaScript en el HTML. Por otra parte, si encapsulamos tu código en una función anónima *autoejecutada*, no podrás acceder a las funciones directamente desde el HTML...
Así que por estos motivos principales, recomendamos el uso de **.addEventListener()**.

Añadir un evento

Veamos como funciona con onclick.

```
<body onclick="cambiarFondo()">
  <h1>Eventos</h1>
  <script>
    function cambiarFondo() {
      let color = "red";
      document.body.style.backgroundColor = color;
      console.log("Nuevo color:", color);
    }
  </script>
</body>
```

Veamos como funciona con addEventListener().

html

```
<h1>addEventListener</h1>
<script src="eventos.js"></script>
```

script

```
window.addEventListener("click", () => {
console.log("Hello? You knocked?");
});
```

Los Event listeners son llamados solo cuando los eventos suceden en el contexto en cual los objetos son registrados

```
<button>Click me</button>
<p>No handler here.</p>
```

```
let button = document.querySelector("button");
button.addEventListener("click", () => {
  console.log("Button clicked.");
});
```

Eventos del Teclado

Cuando se pulsa una tecla del teclado, el navegador lanza un evento "**keydown**".

Cuando se suelta, se produce un evento "**keyup**".

html

```
<p>This page turns violet when you hold the V key.</p>
```

js

```
window.addEventListener("keydown", (event) => {
  if (event.key == "v") {
    document.body.style.background = "violet";
  }
});
window.addEventListener("keyup", (event) => {
  if (event.key == "v") {
    document.body.style.background = "";
  }
});
```

A pesar de su nombre, "keydown" se dispara no sólo cuando la tecla se presiona físicamente hacia abajo. Cuando una tecla se mantiene pulsada, el evento se dispara de nuevo cada vez que la tecla se repita. A veces hay que tener cuidado con esto. Por ejemplo, si se añadiera un botón al DOM cuando se pulsa una tecla y quitarlo de nuevo cuando la cuando se suelta la tecla, podrías añadir accidentalmente cientos de botones cuando la tecla se mantenga pulsada por más tiempo

En el ejemplo anterior el evento se corresponde con la letra. En caso de teclas especiales como “**enter**” la llave (key) será “Enter ”.
Las teclas modificadoras como **shift, control, alt y meta** (command en Mac) generan eventos de teclas al igual que las teclas normales. Pero al buscar combinaciones de teclas, también se puede averiguar si estas teclas se mantienen pulsadas mirando las propiedades **shiftKey**, **ctrlKey**, **altKey** y **metaKey** de los eventos de teclado y ratón.

html

```
<p>Press Control-Space to continue.</p>
```

js

```
window.addEventListener("keydown", (event) => {
  if (event.key == " " && event.ctrlKey) {
    console.log("Continuing!");
  }
});
```

Eventos de mouse

Al pulsar un botón del ratón se disparan varios eventos. Los eventos "**mousedown**" y "**mouseup**" son similares a "**keydown**" y "**keyup**" y se disparan cuando el botón es presionado y liberado. Estos eventos ocurren en los nodos del **DOM** que están inmediatamente debajo del puntero del ratón cuando se produce el evento.
Después del evento "mouseup", se dispara un evento "click" en el nodo más específico que contenga tanto la pulsación como la liberación del botón. Por ejemplo, si yo presiono el botón del ratón en un párrafo y luego muevo el puntero a otro párrafo y suelto el botón, el evento "clic" se producirá en el elemento que contiene ambos párrafos.
Si se producen dos clics juntos, también se producirá un evento "dblclick" (doble clic) se dispara, después del segundo evento de clic.

Cada vez que el puntero del ratón se mueve, se dispara un evento "**mousemove**". Este evento puede utilizarse para seguir la posición del ratón. Una situación común en la que esto es útil es cuando se implementa alguna forma de funcionalidad de arrastre del ratón.
En el siguiente programa se muestra una barra y configura los manejadores de eventos para que al arrastrar a la izquierda o a la derecha en esta barra, se estreche o se haga más ancha:

html

```html
<p>Drag the bar to change its width:</p>
<div style="background: orange; width: 60px; height: 20px">
</div>
```

js

```js
let lastX; // Tracks the last observed mouse X position
let bar = document.querySelector("div");
bar.addEventListener("mousedown", (event) => {
  if (event.button == 0) {
    lastX = event.clientX;
    window.addEventListener("mousemove", moved);
    event.preventDefault(); // Prevent selection
  }
});
function moved(event) {
  if (event.buttons == 0) {
    window.removeEventListener("mousemove", moved);
  } else {
    let dist = event.clientX - lastX;
    let newWidth = Math.max(10, bar.offsetWidth + dist);
    bar.style.width = newWidth + "px";
    lastX = event.clientX;
  }
}
```

mouseenter y mouseleave

Los eventos mouseenter/mouseleave se activan cuando el cursor del mouse entra/sale del elemento.
Las transiciones hacia/desde los descendientes no se cuentan.
Son eventos extremadamente simples.
Cuando el cursor entra en un elemento **mouseenter** se activa. La ubicación exacta del cursor dentro del elemento o sus descendientes no importa.
Cuando el cursor deja el elemento **mouseleave** se activa.
Como puedes ver, los únicos eventos generados son los relacionados con mover el puntero dentro y fuera del elemento superior. No pasa nada cuando el puntero va hacia el descendiente y regresa. Las transiciones entre descendientes se ignoran:

html

```html
<div id="parent" onmouseenter="mouselog(event)" onmouseleave="mouselog(event)">parent
  <div id="child">child</div>
</div>

<textarea id="text"></textarea>
<input type="button" onclick="text.value=''" value="Clear">
```

js

```js
function mouselog(event) {
  let d = new Date();
  text.value += `${d.getHours()}:${d.getMinutes()}:${d.getSeconds()} | ${
    event.type
  } [target: ${event.target.id}]\n`.replace(/(:|^)(\d\D)/, "$10$2");
  text.scrollTop = text.scrollHeight;
}
```

Scroll

Cada vez que se desplaza un elemento, se dispara un evento "**scroll**" sobre él. Esto tiene varios usos tiene varios usos, como saber lo que el usuario está viendo en ese momento.
El siguiente ejemplo dibuja una barra de progreso sobre el documento y la actualiza que se va llenando a medida que se desplaza hacia abajo:

html

```html
<div id="progress"></div>
```

js

```js
document.body.appendChild(document.createTextNode(
"supercalifragilisticexpialidocious ".repeat(1000)));
let bar = document.querySelector("#progress");
window.addEventListener("scroll", () => {
let max = document.body.scrollHeight - innerHeight;
bar.style.width = `${(pageYOffset / max) * 100}%`;
});
```

Removiendo Eventos:

Un nodo sólo puede tener **un** atributo **onclick,** por lo que sólo puede registrar **un** manejador por nodo de esa manera. El método **addEventListener** le permite añadir cualquier número de manejadores, por lo que es seguro añadir manejadores incluso si ya hay ya hay otro manejador en el elemento.
El método **removeEventListener**, llamado con argumentos similares a **addEventListener** elimina un controlador.

html

```
<button>Act-once button</button>
```

js

```
let button = document.querySelector("button");

function once() {
  console.log("Done.");
  button.removeEventListener("click", once);
}
button.addEventListener("click", once);
```

Propagación de eventos

Para la mayoría de los tipos de eventos, los manejadores registrados en nodos con hijos también recibirán los eventos que ocurran en los hijos.
Ejemplo:

```
<div id="parent">
    <div id="children">
        Click
    </div>
</div>
```

En cualquier momento, un controlador de eventos puede llamar al método stopPropagation del objeto del objeto de evento para evitar que los manejadores que están más adelante reciban el evento. Esto puede esto puede ser útil cuando, por ejemplo, usted tiene un botón dentro de otro elemento que se puede pulsar y no quiere que los clics en el botón activen el comportamiento de clic del elemento exterior del elemento exterior.

El siguiente ejemplo registra los manejadores de "mousedown" tanto en un botón como en el párrafo que lo rodea. Cuando se hace clic con el botón derecho del ratón, el manejador del botón llama a botón llama a stopPropagation, lo que evitará que se ejecute el manejador del el párrafo se ejecute. Cuando se pulsa el botón con otro botón del *mouse*, ambos manejadores se ejecutarán.

```
<p>A paragraph with a <button>button</button>.</p>
```

```
let para = document.querySelector("p");
let button = document.querySelector("button");
para.addEventListener("mousedown", () => {
  console.log("Handler for paragraph.");
});
button.addEventListener("mousedown", (event) => {
  console.log("Handler for button.");
  if (event.button == 2) event.stopPropagation();
});
```

La mayoría de los objetos de evento tienen una propiedad de destino que se refiere al nodo donde se originaron. Puedes usar esta propiedad para asegurarte de que no estás manejando accidentalmente que se propaga desde un nodo que no quieres manejar.

También es posible utilizar la propiedad target para lanzar una red amplia para un tipo específico de evento. Por ejemplo, si tienes un nodo que contiene una larga lista de botones puede ser más conveniente registrar un único controlador de clic en el nodo exterior y hacer que utilice la propiedad target para averiguar si se ha pulsado un botón, en lugar de registrar manejadores individuales en todos los botones.

Html

```
<button>A</button>
<button>B</button>
<button>C</button>
```

js

```
document.body.addEventListener("click", (event) => {
  if (event.target.nodeName == "BUTTON") {
    console.log("Clicked", event.target.textContent);
  }
});
```

Acciones por defecto

Muchos **eventos** tienen una acción por defecto asociada a ellos. Si hace clic en un enlace se le llevará al objetivo del enlace. Si pulsa la flecha hacia abajo, el navegador desplazará la página hacia abajo. Si haces clic con el botón derecho, obtendrás un menú contextual. Y así más.

Para la mayoría de los tipos de **eventos**, los manejadores de eventos de **JavaScript** son llamados antes de que el comportamiento por defecto tenga lugar. Si el manejador no quiere que este comportamiento normal,porque ya se ha encargado de manejar el evento, puede llamar al método **preventDefault** en el objeto del evento.

Esto se puede utilizar para implementar sus propios atajos de teclado o menú contextual

También se puede utilizar para interferir de forma odiosa con el comportamiento que los usuarios esperan.

Por ejemplo, aquí hay un enlace que no se puede seguir:

html

```html
<a href="https://developer.mozilla.org/">MDN</a>
```

js

```js
let link = document.querySelector("a");
link.addEventListener("click", (event) => {
  console.log("Nope.");
  event.preventDefault();
});
```

Tratemos de no hacer cosas como estas en las que el usuario está esperando cierto comportamiento al hacer click sobre un enlace, ya que puede tener una mala experiencia al pensar que la página no funciona, pero nos será útil cuando veamos formularios ;)

¡Vamos al código!

Bibliografía consultada y links para seguir investigando

- Jones, D. (2017) ***JavaScript - Novice to ninja*** - United States of America - SitePoint Pty.Ltd
- Gascón Gonzalez U. (2017) ***JavaScript, ¡Inspírate!-*** Leanpub
- Pérez Eguíluz J. (2009) ***Introducción a JavaScript -***Libroweb.es
- Puig Collel J. ***CSS3 y JavaScript avanzado*** - Universidad Oberta de Catalunya
- Rodriguez Jose A. ***Manual de JavaScript*** - Publicado en la página web www.internetmania.net
- ***https://uniwebsidad.com/libros/javascript?from=librosweb***
- ***https://developer.mozilla.org/es/docs/Learn/JavaScript***
- ***https://www.arkaitzgarro.com/javascript/capitulo-15.html#:~:text=Los%20eventos%20de%20JavaScript%20permiten,también%20se%20produce%20un%20evento.***
- ***https://es.javascript.info/mousemove-mouseover-mouseout-mouseenter-mouseleave***

Muchas gracias!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar