Eventos

Introdución

Un evento é unha acción ou ocorrencia que sucede no sistema. Cando ocorre, o sistema lanza un sinal notificando da ocorrencia do evento e proporciona un mecanismo para executar certo código programado para executarse no caso de producirse dito evento. Por exemplo, pode programarse un código para que mostre unha mensaxe informativa cando unha persoa pulse un botón nunha páxina web.

Nunha páxina web poden suceder moitos eventos diferentes, algúns exemplos son:

- Seleccionar, facer clic ou mover o rato sobre un elemento.
- Pulsar unha tecla no teclado
- Redimensionar a ventá do navegador.
- Finalizar a carga da páxina web.
- Enviar un formulario.
- Empezar a ver un vídeo, pausalo ou rematar.

Ademais dos eventos que se poden xerar debido ás accións das persoas usuarias, tamén é posible lanzar eventos dende código, como por exemplo chamando ao método HTMLElement.click() dun elemento.

Para reaccionar a un evento hai que asociarlle un manexador de eventos, que é o bloque de código (normalmente unha función JavaScript) que se executa cando se produce o evento. Ou dito con outras palabras, os elementos DOM poden configurarse para aceptar (“listen” - escoitar) eventos e executar código en resposta (“handler” - manexador do evento).

A interface Event representa un evento que ocorre no DOM. Aínda que hai moitos tipos de eventos e algúns usen outras interfaces diferentes baseadas en Event, todos utilizan os métodos e propiedades comúns da interface Event.

NOTA: ás veces utilízanse indistintamente os nomes manexador/escoitador de eventos (event handler/event listener), aínda que estrictamente falando son distintos: o escoitador escoita cando sucede o evento e o manexador é o código que se executa cando sucede.

Manexadores de eventos

Un manexador de eventos é unha función que se executa cando se produce un evento.

O método recomendado para engadir manexadores de eventos en páxinas web é addEventListener().

EventTarget.addEventListener(type, listener) permite engadir á lista de manexadores unha función que será chamada cando suceda o evento. Os seus parámetros son:

- **type**: nome de evento a escoitar, por exemplo: “click”.
- **listener**: manexador do evento (código a executar cando ocorre o evento).

```
document.getElementById("boton1").addEventListener("click", nomeFuncion);
```

Ligazón a un exemplo: random-color-addeventlistener.html

NOTA: é posible engadir máis dun manexador de eventos a un elemento, incluso para o mesmo evento.

Hai moitos eventos diferentes que pode lanzar un botón. Probar no exemplo anterior a cambiar o evento “click” polos seguintes eventos:

- **focus/blur**: lánzase cando un elemento recibe/perde o foco.
- **dblclick**: lánzase cando se fai dobre clic.
- **mouseover/mouseout:** lánzanse cando o rato pasa por riba/sae do elemento.

O teclado tamén produce eventos cando se pulsan as teclas. A orde da secuencia de eventos é **keydown -> keypress -> keyup**. No caso do rato a orde da secuencia dos seus eventos é: **mousedown -> mouseup -> click.**

Unha vez que se engadiu un manexador de eventos é posible eliminalo usando o método EventTarget.removeEventListener(type, listener).

NOTA: para eliminar un manexador de eventos hai que pasar os mesmos parámetros que se usaron para engadilo, é dicir, hai que pasar o mesmo manexador de eventos, que debe estar almacenado nunha variable. Se se estableceu o manexador usando unha función anónima non hai forma de volver a pasar a mesma función:

```
// forma correcta:
function handler() {
  console.log( '¡Grazas!' );
}

input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

```
elemento.addEventListener( "click" , () => console.log('¡Grazas!'));
// ....
// non se eliminará o manexador de eventos porque son dúas funcións distintas
elemento.removeEventListener( "click", () => console.log('¡Grazas!'));
```

Aínda que existen outras formas de asignar manexadores de eventos, recoméndase usar o método addEventListener. Usar os outros mecanismos considérase unha mala práctica.

Antigamente usábase o atributo **on<Event>** do elemento. Poden asignarse eventos tanto dende HTML como dende JavaScript:

```
<button type="button" id="boton1" onclick="nomeFuncion()" >Botón</button>
```

```
<script>
  function nomeFuncion() {...}
  elemento.onclick = nomeFuncion;
</script>
```

Observar que en HTML se utiliza a chamada á función en lugar da definición da función. Isto é así porque cando o navegador le o atributo tradúceo a JavaScript creando unha función manexadora co contido do atributo:

```
button.onclick = function() {
  nomeFuncion(); // <-- o contido do atributo vai aquí
};
```

O problema de asignar manexadores de eventos usando o atributo on<Event> é que non é posible asignar múltiples manexadores ao mesmo evento. Para isto xurdiu a alternativa **recomendada actualmente,** de usar os métodos especiais addEventListener() e removeEventListener().

Ademais, hai algúns eventos que só funcionan usando addEventListener(), como é o evento DOMContenetLoaded, que se activa cando o documento está cargado e o DOM está construído:

```
document.addEventListener("DOMContentLoaded", function() {
  console.log("DOM construido");
});
```

NOTA: recordar que, antes de asignar o manexador de eventos, hai que asegurarse de que o elemento da árbore DOM está creado.

Obxecto do evento

Cando ocorre un evento, o navegador crea un obxecto evento que contén información relativa ao mesmo. Este obxecto pásase automaticamente como argumento ao manexador do evento.

```
function handler(event) {
  console.log( “Elemento que produciu o evento “ + event.target );
}
input.addEventListener("click", handler);
```

Aínda que se pode poñer calquera nome ao parámetro da función, recoméndase usar os nomes **event**, **evt** ou **e**.

A maioría dos obxectos evento teñen un conxunto de propiedades e métodos estándar dispoñibles, que se poden consultar na documentación de Event, por exemplo:

- **event.type**: tipo de evento, por exemplo **click.**
- **event.target**: identifica o elemento onde ocorre o evento ou onde se iniciou o evento.
- **event.currentTarget**: refírese ao elemento no foi engadido o manexador de eventos. Pode non coincidir co event.target, pois o evento puido producirse nun elemento descendente e propagarse pola árbore DOM.
- **event.clientX/event.clientY**: coordenadas do cursor relativas á ventá, para eventos de cursor.

Hai máis propiedades e moitas delas dependen do tipo de evento. Así, os eventos de teclado (keydown) teñen un conxundo de propiedades e os de cursor teñen outras. O evento KeyboardEvent, por exemplo, ten a propiedade **key** que informa de que tecla se pulsou.

```
<input id="textBox" type="text" />
<div id="output"></div>
```

```
const textBox = document.querySelector("#textBox");
const output = document.querySelector("#output");
textBox.addEventListener('keydown', (event) => output.textContent = `You pressed ${event.key}".`);
```

Exercicios:

1. Dado o seguinte código HTML:

   ```
   <input type="button" id="ocultar" value="Fai clic para que desapareza o texto" /><br><br>
   <input type="text" id="textoExercicio1" name="texto" /><br><br>
   <div id="texto">Texto</div>
   ```

   Engade os eventos que se indican:

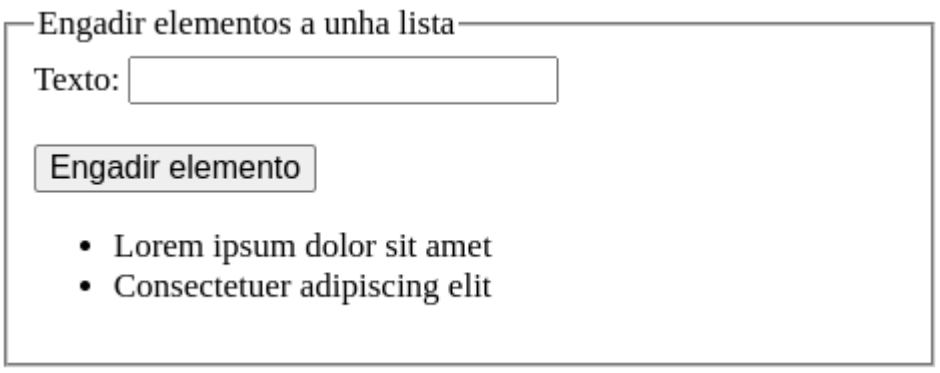
   - Cando o cursor do rato entre e saia do botón, mostra unha mensaxe por consola indicándoo.
   - Ao pulsar o botón debe desaparecer o div con id=texto.
   - Cando se escriba algo na caixa de texto, debe mostrarse a tecla pulsada no div e tamén o código da tecla pulsada. Ademais, se o div estaba oculto, debe mostrarse.

2. Dado o seguinte código HTML, engade a configuración CSS para que inicialmente o span estea oculto. Ao pulsar a ligazón debe facerse visible o contido do span e desaparecer a ligazón.

```
<p id="textoExercicio2">
   Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
   <span id="adicional" class="oculto">Nullam sit amet nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis scelerisque. Donec lacus neque, vehicula in, eleifend vitae, venenatis ac, felis.</span>
</p>

<a id="ligazon" href="#">Seguir lendo...</a>
```

3. Crea unha páxina web que teña unha caixa de texto un botón e unha lista. Fai que ao pulsar o botón se engada o texto da caixa de texto como novo elemento da lista. Inicialmente, a páxina debería mostrar algo similar á seguinte imaxe:



4. Nunha páxina HTML hai un botón ao que fai referencia a variable button do seguinte código. Indica que manexadores de eventos se executan dos seguintes:

```
button.addEventListener("click", () => console.log("1"));
button.removeEventListener("click", () => console.log("1"));
button.onclick = () => console.log(2);
```

5. Crea un menú que se abra/colapse ao facer clic. Inicialmente o menú debe estar colapsado e ao pulsar no texto “Sweeties (click me)!” deben mostrarse as opcións do menú.

Sweeties (click me)!	Sweeties (click me)! Cake Donut Honey

Debes asegurarte que a funcionalidade só se habilita cando se pulsa sobre o texto “Sweeties (click me)!” e non sobre calquera outra zona da páxina/liña.

Nestes casos é útil modificar o cursor cando pase por riba do texto para que teña estilo “pointer” e informar á persoa usuaria que esa é unha zona na que se pode pulsar.

Evitar comportamento por defecto

As accións realizadas pola persoa usuaria nunha páxina web e os seus eventos asociados teñen un comportamento por defecto. Así, por exemplo, cando se pulsa nunha ligazón cargarase no navegador a páxina web da nova ligazón.

Algunhas veces interesa evitar a acción por defecto e executar outro código. Por exemplo, cando se pulsa o botón de enviar nun formulario, por defecto envíanse os datos á páxina especificada no servidor para o seu procesamento. O problema aparece cando hai erros no formulario, sendo necesario mostrar unha mensaxe de advertencia para que se corrixan, antes de enviar os datos ao servidor.

No caso de que se queira evitar o comportamento por defecto utilízase o método preventDefault(). Ligazón a exemplo onde se comproba se os campos do formulario están baleiros, mostrando unha mensaxe en caso de que sexa certo e evitando o envío do formulario.

Propagación de eventos

A propagación de eventos describe como xestiona o navegador os eventos en elementos aniñados.

O seguinte exemplo ten un botón dentro dun contedor . ¿Que sucede cando se engade un manexador de eventos no elemento pai e se pulsa o botón?

```
<div id="container">
  <button>Click me!</button>
</div>
<pre id="output"></pre>
```

```
const output = document.querySelector('#output');
function handleClick(e) {
  output.textContent += `You clicked on a ${e.currentTarget.tagName} element\n`;
}

const container = document.querySelector('#container');
container.addEventListener('click', handleClick);
```

```
You clicked on a DIV element
```

Se se proba o código anterior comprobarase que o pai lanza o evento. Isto ten sentido, porque o botón está dentro do , polo que se se pulsa o botón tamén se está pulsando implicitamente o elemento div.

¿Que pasará se se engaden manexadores de eventos ao body, ao div e ao botón?

```
<body>
  <div id="container">
    <button>Click me!</button>
  </div>
  <pre id="output"></pre>
</body>
```

```
const output = document.querySelector('#output');
function handleClick(e) {
  output.textContent += `You clicked on a ${e.currentTarget.tagName} element\n`;
}

const container = document.querySelector('#container');
const button = document.querySelector('button');

document.body.addEventListener('click', handleClick);
container.addEventListener('click', handleClick);
button.addEventListener('click', handleClick);
```

```
You clicked on a BUTTON element
You clicked on a DIV element
You clicked on a BODY element
```

Co exemplo anterior pode comprobarse que os 3 elementos lanzan o evento cando se pulsa o botón. Primeiro lánzase o evento do botón, despois o do div e despois o do body.

Cando ocorre un evento, dise que este se **propaga** dende o elemento máis interno aos ancestros. Primeiro execútanse os manexadores de eventos do elemento máis interno e despois os dos ancestros. Este comportamento pode ser útil, sen embargo tamén pode causar comportamentos inesperados.

NOTA: case todos os eventos se propagan. Hai algunhas excepcións, como o evento **focus**, que non se propaga, aínda que a maioría si que o fai.

O seguinte exemplo contén un vídeo, que está oculto inicialmente, e un botón “Display video”. Preténdese que cando se pulse o botón se mostre o div co vídeo e cando se pulse sobre o vídeo este empece a reproducirse. Ademais, cando se pulse sobre o div, este debe ocultarse:

```
<style>
  .hidden { display: none;}
</style>
```

```
<button>Display video</button>
<div class="hidden">
  <video>
    <source
src="https://raw.githubusercontent.com/mdn/learning-area/master/javascript/building-blocks/events/rabbit320.webm" type="video/webm" />
    <p>Your browser doesn't support HTML video.</p>
  </video>
</div>
```

```
const btn = document.querySelector('button');
const box = document.querySelector('div');
const video = document.querySelector('video');

btn.addEventListener('click', () => box.classList.remove('hidden'));
video.addEventListener('click', () => video.play());
box.addEventListener('click', () => box.classList.add('hidden'));
```

Observa que cando se pulsa o botón móstrase o div co vídeo. Sen embargo, cando se pulsa o vídeo, o vídeo comeza a reproducirse e a caixa ocúltase.

Isto é así porque o vídeo está dentro do , polo que ao pulsar sobre o vídeo lánzanse os dous eventos (o do vídeo e o do div).

Para evitar a propagación do evento neste caso pode usarse o método stopPropagation(), que ao executarse evita a propagación do evento a outros elementos. Habería que cambiar o manexador de eventos do video como se indica a continuación:

```
video.addEventListener('click', (event) => {
  event.stopPropagation();
  video.play();
});
```

NOTA: A propagación é conveninte e necesaria, polo que non é recomendable detela sen que exista un motivo xustificado.

Delegación de eventos

No apartado anterior viuse o problema da propagación de eventos. Sen embargo, non sempre é un problema, ás veces é moi útil. Por exemplo, cando se quere executar o mesmo código ao interactuar cunha serie de elementos e os seus fillos. Neste caso bastará con engadir o manexador do evento ao elemento pai en lugar de ter que engadilo a cada un dos fillos.

Ligazón a un exemplo que usa a delegación de eventos. Este exemplo divide a páxina en 16 áreas e cada vez que se pulsa sobre un área cámbiase a cor de fondo da mesma. En lugar de engadir 16 manexadores de eventos asígnase só un manexador ao elemento pai. Para saber a área na que se pulsou utilízase a propiedade **event.target**. Se se quixese acceder ao elemento que ten asignado o manexador de eventos usaríase **event.currentTarget**.

¿Que pasaría se no exemplo anterior as celas das táboas teñen contido HTML?. Por exemplo, ¿que pasaría se unha cela ten o seguinte código?

```
<div class="tile">
    <p>Lorem ipsum.</p>
</div>
```

Se se proba a executar o exemplo modificando unha cela da táboa polo código anterior comprobarase que cando se pulsa sobre o elemento <p>, cámbiase a cor de fondo do parágrafo. Para que se cambiase a cor de fondo da cela, habería que navegar pola árbore DOM ata obter o div relativo á cela. Isto pode obterse co método Element.closest(selector) que navega pola árbore DOM dende o elemento aos ancestros para buscar un nodo que encaixe co selector pasado como parámetro.

```
container.addEventListener("click", (event) => {
    let tile = event.target.closest("div");
    tile.style.backgroundColor = bgChange();
});
```

Captura de eventos

A captura de eventos prodúcese na orde inversa á propagación. É dicir, o evento prodúcese primeiro no elemento máis externo e vaise lanzando polos elementos aniñados ata o máis interno.

Por defecto a captura de eventos está deshabilitada. Para habilitala hai que pasar un parámetro no método addEventListener().

O seguinte exemplo é o mesmo que o usado na propagación, excepto que o parámetro capture está activado:

```
<body>
  <div id="container">
    <button>Click me!</button>
  </div>
  <pre id="output"></pre>
</body>
```

```
const output = document.querySelector('#output');
function handleClick(e) {
  output.textContent += `You clicked on a ${e.currentTarget.tagName} element\n`;
}

const container = document.querySelector('#container');
const button = document.querySelector('button');

document.body.addEventListener('click', handleClick, { capture: true });
container.addEventListener('click', handleClick, { capture: true });
// por razóns históricas pode poñerse simplemente true
// equivalente a container.addEventListener('click', handleClick, true);
button.addEventListener('click', handleClick);
```

```
You clicked on a BODY element
You clicked on a DIV element
You clicked on a BUTTON element
```

Observar que a orde das mensaxe é inversa á que ocorre na propagación.

Antigamente, Nestcape só usaba a captura de eventos e Internet Explorer usaba a propagación. Cando o W3C decidiu estandarizar este comportamento e se alcanzou un consenso, chegouse a este sistema que incluía a propagación e a captura, que é o sistema que implementan os navegadores actuais.

NOTA: a maioría das veces os manexadores de eventos son rexistrados na fase de propagación.

Exercicios:

1. Descarga o código HTML e CSS e engade o JavaScript necesario para que ao pulsar no botón [X] se elimine a mensaxe correspondente.

   Debes engadir un único *listener* e só se deben eliminar as mensaxes ao pulsar [X].

2. Descarga o código HTML e realiza as seguintes operacións con JavaScript:
   a. Utilizando **un só manexador de eventos** fai que ao pulsar sobre un nodo da árbore se mostren/oculten os nodos fillos.
   b. ¿Como farías para que só funcionase ao pulsar no título e non en calquera espacio baleiro da liña?. Impleméntao en JavaScript.

3. Descarga o código HTML e fai que a táboa se poida ordenar. Cando se pulse nun <th>, deberían ordenarse os datos pola columna sobre a que se fixo clic.

   Cada columna ten o tipo de datos no atributo **data-type**, neste caso unha columna ten números e outra strings. Hai que ter en conta o tipo de datos na ordenación.

   O código programado debe funcionar con táboas máis grandes, tanto en número de filas como de columnas.

   Utiliza un só manexador de eventos.

4. Descarga o código HTML e fai que ao pulsar en calquera ligazón do elemento con id "contents", se pida confirmación antes de redirixir ao novo enderezo. Se a persoa pulsa cancelar na confirmación, non se navegará á nova páxina.

   Utiliza un só manexador de eventos.

   Fíxate que as ligazóns poden ter etiquetas aniñadas "..."

5. Descarga o código HTML e CSS e fai que ao pulsar sobre unha imaxe en miniatura se cargue na imaxe principal a versión grande da imaxe pulsada. Actualiza tamén o atributo "alt" da imaxe principal co título da nova imaxe cargada.

   Utiliza un só manexador de eventos.

6. Descarga o código HTML e programa a funcionalidade para que, con un único manexador de eventos, cando se seleccione un elemento da lista se lle engada a clase CSS "selected" e se lle quite esta clase a todos os demais elementos da lista.

Referencias

Para a elaboración deste material utilizáronse, entre outros, os recursos que se enumeran a continuación:

- JavaScript | MDN
- Client-side web APIs - Learn web development | MDN
- JavaScript Tutorial - w3schools
- Desarrollo Web en Entorno Cliente | materials
- Javascript en español - Lenguaje JS
- The Modern JavaScript Tutorial
- Eloquent JavaScript
- Introduction to events - Learn web development | MDN