3 de julio de 2023

Introducción a los eventos en el navegador

Un evento es una señal de que algo ocurrió. Todos los nodos del DOM generan dichas señales (pero los eventos no están limitados sólo al DOM).

Aquí hay una lista con los eventos del DOM más utilizados, solo para echar un vistazo:

Eventos del mouse:

- `click` – cuando el mouse hace click sobre un elemento (los dispositivos touch lo generan con un toque).
- `contextmenu` – cuando el mouse hace click derecho sobre un elemento.
- `mouseover` / `mouseout` – cuando el cursor del mouse ingresa/abandona un elemento.
- `mousedown` / `mouseup` – cuando el botón del mouse es presionado/soltado sobre un elemento.
- `mousemove` – cuando el mouse se mueve.

Eventos del teclado:

- `keydown` / `keyup` – cuando se presiona/suelta una tecla.

Eventos del elemento form:

- `submit` – cuando el visitante envía un `<form>` .
- `focus` – cuando el visitante hace foco en un elemento, por ejemplo un `<input>` .

Eventos del documento:

- `DOMContentLoaded` --cuando el HTML es cargado y procesado, el DOM está completamente construido

Eventos del CSS:

- `transitionend` – cuando una animación CSS concluye.

Hay muchos más eventos. Entraremos en más detalles con eventos particulares en los siguientes capítulos.

Controladores de eventos

Para reaccionar a los eventos podemos asignar un *handler (controlador)* el cual es una función que se ejecuta en caso de un evento.

Los handlers son una forma de ejecutar código JavaScript en caso de acciones por parte del usuario.

Hay muchas maneras de asignar un handler. Vamos a verlas empezando por las más simples.

Atributo HTML

Un handler puede ser establecido en el HTML con un atributo llamado `on<event>` .

Por ejemplo, para asignar un handler `click` a un `input` podemos usar `onclick` , como aquí:

```
<input value="Haz click aquí" onclick="alert('¡Click!')" type="button">
```

Al hacer click, el código dentro de `onclick` se ejecuta.

Toma en cuenta que dentro de `onclick` usamos comillas simples, porque el atributo en sí va entre comillas dobles. Si olvidamos que el código está dentro del atributo y usamos comillas dobles dentro, así: `onclick="alert("Click!")"` , no funcionará correctamente.

Un atributo HTML no es un lugar conveniente para escribir un montón de código, así que mejor creamos una función JavaScript y la llamamos allí.

Aquí un click ejecuta la función `countRabbits()` :

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Conejo número " + i);
    }
  }
</script>

<input type="button" onclick="countRabbits()" value="¡Cuenta los conejos!">
```

¡Cuenta los conejos!

Como sabemos, los nombres de los atributos HTML no distinguen entre mayúsculas y minúsculas, entonces `ONCLICK` funciona bien al igual que `onClick` y `onCLICK` … Pero usualmente los atributos van con minúsculas: `onclick` .

Propiedad del DOM

Podemos asignar un handler usando una propiedad del DOM `on<event>` .

Por ejemplo, `elem.onclick` :

```
<input id="elem" type="button" value="Haz click en mí">
<script>
  elem.onclick = function() {
    alert('¡Gracias!');
  };
</script>
```

Haz click en mí

Si el handler es asignado usando un atributo HTML entonces el navegador lo lee, crea una nueva función desde el contenido del atributo y lo escribe en la propiedad del DOM.

Esta forma en realidad es la misma que ya habíamos visto antes.

Estás dos piezas de código funcionan igual:

1. Solo HTML:

```
<input type="button" onclick="alert('¡Click!')" value="Botón">
```

Botón

2. HTML + JS:

```
<input type="button" id="button" value="Botón">
<script>
  button.onclick = function() {
    alert('¡Click!');
  };
</script>
```

Botón

En el primer ejemplo el atributo HTML es usado para inicializar el `button.onclick` , mientras que en el segundo ejemplo se usa el script. Esa es toda la diferencia.

Como solo hay una propiedad `onclick` , no podemos asignar más de un handler.

En el siguiente ejemplo se agrega un handler con JavaScript que sobrescribe el handler existente:

```
<input type="button" id="elem" onclick="alert('Antes')" value="¡Haz click en mi
<script>
  elem.onclick = function() { // sobrescribe el handler existente
    alert('Después'); // solo se mostrará este
  };
</script>
```

¡Haz click en mí!

Para eliminar un handler, asigna `elem.onclick = null` .

Accediendo al elemento: this

El valor de `this` dentro de un handler es el elemento, el cual tiene el handler dentro.

En el siguiente código el `button` muestra su contenido usando `this.innerHTML` :

```
<button onclick="alert(this.innerHTML)">Haz click en mí</button>
```

Haz click en mí

Posibles errores

Si estás empezando a trabajar con eventos, por favor, nota algunas sutilezas.

Nosotros podemos establecer una función existente como un handler:

```
function sayThanks() {
  alert('¡Gracias!');
}

elem.onclick = sayThanks;
```

Pero ten cuidado: la función debe ser asignada como `sayThanks` , no `sayThanks()` .

```
// correcto
button.onclick = sayThanks;

// incorrecto
button.onclick = sayThanks();
```

Si agregamos paréntesis, `sayThanks()` se convierte en una llamada de función. En ese caso la última linea toma el *resultado* de la ejecución de la función, que es `undefined` (ya que la función no devuelve nada), y lo asigna a `onclick` . Esto no funciona.

...Por otro lado, en el markup necesitamos los paréntesis:

```
<input type="button" id="button" onclick="sayThanks()">
```

La diferencia es fácil de explicar. Cuando el navegador lee el atributo crea una función handler con cuerpo a partir del contenido del atributo.

Por lo que el markup genera esta propiedad:

```
button.onclick = function() {
  sayThanks(); // <-- el contenido del atributo va aquí
};
```

No uses `setAttribute` para handlers.

Tal llamada no funcionará:

```
// un click sobre <body> generará errores,
// debido a que los atributos siempre son strings, la función se convierte en u
document.body.setAttribute('onclick', function() { alert(1) });
```

Las mayúsculas en las propiedades DOM importan.

Asignar un handler a `elem.onclick`, en lugar de `elem.ONCLICK`, ya que las propiedades DOM son sensibles a mayúsculas.

addEventListener

El problema fundamental de las formas ya mencionadas para asignar handlers es que *no podemos asignar multiples handlers a un solo evento*.

Digamos que una parte de nuestro código quiere resaltar un botón al hacer click, y otra quiere mostrar un mensaje en el mismo click.

Nos gustaría asignar dos handlers de eventos para eso. Pero una nueva propiedad DOM sobrescribirá la que ya existe:

```
input.onclick = function() { alert(1); }
// ...
input.onclick = function() { alert(2); } //  el handler reemplaza el handler ar
```

Los desarrolladores de estándares de la web entendieron eso hace mucho tiempo y sugirieron una forma alternativa de administrar los handlers utilizando los métodos especiales `addEventListener` y `removeEventListener`, que no tienen este problema.

La sintaxis para agregar un handler:

```
element.addEventListener(event, handler, [options]);
```

`event`

Nombre del evento, por ejemplo: `"click"`.

`handler`

La función handler.

`options`

Un objeto adicional, opcional, con las propiedades:

- `once` : si es `true` entonces el listener se remueve automáticamente después de activarlo.
- `capture` : la fase en la que se controla el evento, que será cubierta en el capítulo Propagación y captura. Por razones históricas, `options` también puede ser `false/true` , lo que es igual a `{capture: false/true}` .
- `passive` : si es `true` entonces el handler no llamará a `preventDefault()` , esto lo explicaremos más adelante en Acciones predeterminadas del navegador.

Para remover el handler, usa `removeEventListener` :

```
element.removeEventListener(event, handler, [options]);
```

Remover requiere la misma función

Para remover un handler deberemos pasar exactamente la misma función que asignamos.

Esto no funciona:

```
elem.addEventListener( "click" , () => alert('¡Gracias!'));
// ....
elem.removeEventListener( "click", () => alert('¡Gracias!'));
```

El handler no será removido porque `removeEventListener` obtiene otra función, con el mismo código, pero eso no importa, ya que es un objeto de función diferente.

Aquí está la manera correcta:

```
function handler() {
  alert( '¡Gracias!' );
}

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

Por favor nota que si no almacenamos la función en una variable entonces no podremos removerla. No hay forma de "volver a leer" los handlers asignados por `addEventListener` .

Múltiples llamadas a `addEventListener` permiten agregar múltiples handlers:

```
<input id="elem" type="button" value="Haz click en mí"/>

```

```
<script>
  function handler1() {
    alert('¡Gracias!');
  };

  function handler2() {
    alert('¡Gracias de nuevo!');
  }

  elem.onclick = () => alert("Hola");
  elem.addEventListener("click", handler1); // Gracias!
  elem.addEventListener("click", handler2); // Gracias de nuevo!
</script>
```

Como podemos ver en el ejemplo anterior, podemos establecer handlers *tanto* usando un propiedad DOM como `addEventListener` juntos. Pero por lo general solo usamos una de esas maneras.

Para algunos eventos, los handlers solo funcionan con `addEventListener`

Hay eventos que no pueden ser asignados por medio de una propiedad del DOM, sino solamente con `addEventListener`.

Por ejemplo, el evento `DOMContentLoaded`, que se activa cuando el documento está cargado y el DOM está construido.

```
// nunca se ejecutará
document.onDOMContentLoaded = function() {
  alert("DOM construido");
};
```

```
// Así sí funciona
document.addEventListener("DOMContentLoaded", function() {
  alert("DOM construido");
});
```

Por lo que `addEventListener` es más universal. Aún así, tales eventos son una excepción más que la regla.

Objeto del evento

Pero para manejar correctamente un evento necesitamos saber todavía más acerca de lo que está pasando. No solo si fue un "click" o un "teclazo", sino ¿cuáles eran coordenadas del cursor, o qué tecla fue oprimida? Y así.

Cuando un evento ocurre, el navegador crea un *objeto del evento*, coloca los detalles dentro y los pasa como un argumento al handler.

Aquí hay un ejemplo para obtener las coordenadas del cursor a partir del objeto del evento:

```
<input type="button" value="¡Haz click en mí!" id="elem">

<script>
  elem.onclick = function(event) {
    // muestra el tipo de evento, el elemento y las coordenadas del click
    alert(event.type + " en el " + event.currentTarget);
    alert("Coordenadas: " + event.clientX + ":" + event.clientY);
  };
</script>
```

Algunas propiedades del objeto `event`:

`event.type`

Tipo de evento, en este caso fue `"click"`.

`event.currentTarget`

Elemento que maneja el evento. Lo que exactamente igual a `this`, a menos que el handler sea una función de flecha o su `this` esté vinculado a otra cosa, entonces podemos obtener el elemento desde `event.currentTarget`.

`event.clientX / event.clientY`

Coordenadas del cursor relativas a la ventana, para eventos de cursor.

Hay más propiedades. Muchas de ellas dependen del tipo de evento: los eventos del teclado tienen un conjunto de propiedades, y las de cursor, otro. Los estudiaremos después, cuando lleguemos a los detalles de diferentes eventos.

El objeto del evento también está disponible para handlers HTML

Si asignamos un handler en HTML también podemos usar el objeto `event`, así:

```
<input type="button" onclick="alert(event.type)" value="Event type">
```

<input type="button" value="Event type">

Esto es posible porque cuando el navegador lee el atributo, crea un handler como este: `function(event) { alert(event.type) }`. Lo que significa que el primer argumento es llamado `"event"` y el cuerpo es tomado del atributo.

Objetos handlers: handleEvent

Podemos asignar no solo una función, sino un objeto como handler del evento usando `addEventListener`. Cuando el evento ocurre, el método `handleEvent` es llamado.

Por ejemplo:

```
<button id="elem">Haz click en mí</button>

<script>
  let obj = {
    handleEvent(event) {
      alert(event.type + " en " + event.currentTarget);
    }
  };

  elem.addEventListener('click', obj);
</script>
```

Como podemos ver, cuando `addEventListener` recibe como handler a un objeto, llama a `obj.handleEvent(event)` en caso de un evento.

También podemos usar objetos de una clase personalizada:

```
<button id="elem">Haz click en mí</button>

<script>
  class Menu {
    handleEvent(event) {
      switch(event.type) {
        case 'mousedown':
          elem.innerHTML = "Botón del mouse presionado";
          break;
        case 'mouseup':
          elem.innerHTML += "...y soltado.";
          break;
      }
    }
  }

  let menu = new Menu();

  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
```

Aquí el mismo objeto maneja ambos eventos. Nota que necesitamos configurar explícitamente los eventos a escuchar usando `addEventListener`. El objeto `menu` solo obtiene `mousedown` y `mouseup` aquí, no hay ningún otro tipo de eventos.

El método `handleEvent` no tiene que hacer todo el trabajo por sí solo. En su lugar puede llamar a otros métodos específicos de eventos, como este:

```
<button id="elem">Haz click en mí</button>

```

```
<script>
  class Menu {
    handleEvent(event) {
      // mousedown -> onMousedown
      let method = 'on' + event.type[0].toUpperCase() + event.type.slice(1);
      this[method](event);
    }

    onMousedown() {
      elem.innerHTML = "Botón del mouse presionado";
    }

    onMouseup() {
      elem.innerHTML += "...y soltado.";
    }
  }

  let menu = new Menu();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
```

Ahora los handlers del evento están claramente separados, lo que puede ser más fácil de mantener.

Resumen

Hay tres formas de asignar handlers:

1. Atributos HTML: `onclick="..."` .
2. Propiedades del DOM: `elem.onclick = function` .
3. Métodos: `elem.addEventListener(event, handler[, phase])` para agregar ó `removeEventListener` para remover.

Los atributos HTML se usan con moderación, porque JavaScript en medio de una etiqueta HTML luce un poco extraño y ajeno. Además no podemos escribir montones de código ahí.

Las propiedades del DOM son buenas para usar, pero no podemos asignar más de un handler a un evento en particular. En la mayoría de casos esta limitación no es apremiante.

La última forma es la más flexible, pero también es la más larga para escribir. Unos pocos eventos solo funcionan con ésta, por ejemplo `transitionend` y `DOMContentLoaded` (que veremos después). Además `addEventListener` soporta objetos como handlers de eventos. En este caso `handleEvent` es llamado en caso del evento.

No importa como asignes el handler, este obtiene un objeto como primer argumento. Este objeto contiene los detalles sobre lo que pasó.

Vamos a aprender más sobre eventos en general y sobre diferentes tipos de eventos en los siguientes capítulos.

Tareas

Ocultar con un click

importancia: 5

Agrega JavaScript al `button` para hacer que `<div id="text">` desaparezca al clickearlo.

El demo:

Haz click para desaparecer el texto
Texto

Abrir un entorno controlado para la tarea.

solución

Ocultarse

importancia: 5

Crea un botón que se oculte a sí mismo al darle un click.

Así: Click para esconder

solución

¿Qué handlers se ejecutan?

importancia: 5

Hay un botón en la variable. No hay handlers en él.

¿Qué handlers se ejecutan con el click después del siguiente código? ¿Qué alertas se muestran?

```
1  button.addEventListener("click", () => alert("1"));
2
3  button.removeEventListener("click", () => alert("1"));
4
5  button.onclick = () => alert(2);
```

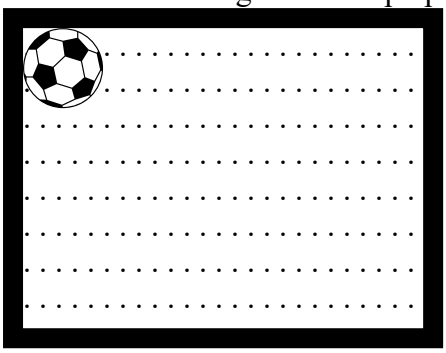
solución

Mueve el balón por el campo

importancia: 5

Mueve el balón por el campo con un click. Así:

Haz click en un lugar del campo para mover el balón allí.



Requerimientos:

- El centro del balón debe quedar exactamente bajo el cursor al hacer click (sin atravesar el borde del campo si es posible).
- Las animaciones CSS son bienvenidas.
- El balón no debe cruzar los límites del campo.
- Cuando la página se desplace nada se debe romper.

Notas:

- El código también debe funcionar con medidas diferentes de campo y balón, no debe estar asociado a ningún valor fijo.
- Usa las propiedades `event.clientX/event.clientY` para las coordenadas del click.

Abrir un entorno controlado para la tarea.

solución

Crear un menú deslizante

importancia: 5

Crea un menú que se abra/colapse al hacer click:

▶ Sweeties (click me)!

P.D. El HTML/CSS del documento fuente se debe modificar.

Abrir un entorno controlado para la tarea.

solución

Agregar un botón de cierre

importancia: 5

Hay una lista de mensajes.

Usa JavaScript para agregar un botón de cierre en la esquina superior derecha de cada mensaje.

El resultado debería verse algo así:

Horse [x]

The horse is one of two extant subspecies of Equus ferus. It is an odd-toed ungulate mammal belonging to the taxonomic family Equidae. The horse has evolved over the past 45 to 55 million years from a small multi-toed creature, Eohippus, into the large, single-toed animal of today.

Donkey [x]

The donkey or ass (Equus africanus asinus) is a domesticated member of the horse family, Equidae. The wild ancestor of the donkey is the African wild ass, E. africanus. The donkey has been used as a working animal for at least 5000 years.

Cat [x]

The domestic cat (Latin: Felis catus) is a small, typically furry, carnivorous mammal. They are often called house cats when kept as indoor pets or simply cats when there is no need to distinguish them from other felids and felines. Cats are often valued by humans for companionship and for their ability to hunt vermin.

Abrir un entorno controlado para la tarea.

solución

Carrusel

importancia: 4

Crea un "carrusel": una cinta de imágenes que se puede desplazar haciendo clic en las flechas.

Más adelante podemos agregarle más funciones: desplazamiento infinito, carga dinámica, etc.

P.D. Para esta tarea, la estructura HTML / CSS es en realidad el 90% de la solución.

Abrir un entorno controlado para la tarea.

solución

Lección anterior Próxima lección

Compartir

Mapa del Tutorial

Comentarios

- Si tiene sugerencias sobre qué mejorar, por favor enviar una propuesta de GitHub o una solicitud de extracción en lugar de comentar.
- Si no puede entender algo en el artículo, por favor explique.
- Para insertar algunas palabras de código, use la etiqueta `<code>`, para varias líneas – envolverlas en la etiqueta `<pre>`, para más de 10 líneas – utilice una entorno controlado (sandbox) (plnkr, jsbin, codepen...)

acerca del proyecto contáctenos