UT 6 - 1 - EVENTOS - Introducción (Resumen)

Contenido

[1. Introducción a los eventos en el navegador 2](#_Toc121727480)

[Lista de los eventos del DOM más utilizados 2](#_Toc121727481)

[Controladores de eventos 2](#_Toc121727482)

[Objeto del evento 4](#_Toc121727483)

[~~Objeto del evento (AMPLIACIÓN)~~ 4](#_Toc121727484)

[2. Propagación y captura 5](#_Toc121727485)

[Fase de captura y fase de burbuja 5](#_Toc121727486)

[event.target 5](#_Toc121727487)

[Detener la propagación 5](#_Toc121727488)

[3. Un patrón de manejo de eventos: delegación de eventos 7](#_Toc121727489)

[4. Acciones predeterminadas del navegador 7](#_Toc121727490)

[~~5. Envío de eventos personalizados~~ 8](#_Toc121727491)

# 1. Introducción a los eventos en el navegador

## Lista de los eventos del DOM más utilizados

Es una lista incompleta, para empezar a utilizar los eventos.

**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 sobre un elemento.

**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 se centra sobre 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.

## Controladores de eventos

Cómo asignar una handler (controlador) para que se ejecute cuando se produzca un evento.

Hay varias formas de hacerlo.

**Atributo HTML**

* Asignar un manejador de eventos: Atributo on<event> en el HTML

Ejemplo:

<input id="el1" onclick = "alert('hola');" type="button">

* Eliminar un manejador de eventos: no se puede

**Propiedad del DOM**

* Asignar un manejador de eventos: método elem.on<event>

Ejemplo:

elemento.onclick = funcion() {

alert("Hola");

}

* Eliminar un manejador de eventos: Asignar null al método elem.on<event>

Ejemplo:

elemento.onclick = null;

Con los dos métodos anteriores, sólo se puede asignar un handler para cada evento.

this es el elemento que tiene el handler.

Posibles errores:

* Cuidado con los paréntesis, si utilizamos la propiedad del DOM
* En JS se distinguen mayúsculas y minúsculas; en HTML (atributos del elemento), no.
* No se puede utilizar setAttribute() para handlers

**addEventListener**

* Asignar un manejador de eventos: elemento.addEventListener().

Sintaxis:

elem.addEventListener(evento, handler, [options])

* + evento: nombre del evento (cadena)
  + handler: puntero a la función que manejará el evento
  + [options]: objeto adicional, opcional, o boolean
* Eliminar un manejador de eventos: elemento.removeEventListener()

Sintaxis:

elem.removeEventListaner(evento, handler, [options])

Ejemplo:

function handler() {

alert( '¡Gracias!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);

* Con este método, se pueden asignar varios handler para cada evento.
* Algunos eventos sólo se pueden gestionar con este método (ej. "DOMContentLoaded"

## Objeto del evento

Cuando se produce un evento, el navegador crea un objeto asociado al evento, coloca los detalles dentro y lo pasa como argumento al handler.

Ejemplo1:

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

Ejemplo2:

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

<script>

elem.onclick = function(event) {

// muestra el tipo de evento y las coordenadas del click

alert(event.type + " en el " + event.currentTarget);

alert("Coordenadas: " + event.clientX + ":" + event.clientY);

};

</script>

Algunas propiedades/métodos del objeto event:

**event.type**

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

**event.target**

Elemento en el que se ha hecho 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.

**event.stopPropagation()**

Impide la propagación del evento (se verá más adelante).

**event.preventDefault()**

Impide la acción por defecto asociada al evento (se verá más adelante).

## ~~Objeto del evento (AMPLIACIÓN)~~

Para manejar un evento que se produce en un elemento, hemos visto que con addEventListener se puede asignar una función al evento.

Con addEventListener también se puede asignar un objeto que tenga un método hadleEvent().

# 2. Propagación y captura

## Fase de captura y fase de burbuja

Cuando se produce un evento sobre un elemento, este afecta a los "ascendientes de ese elemento".

Casi todos los eventos se propagan (ej. focus no se propaga)

El estándar de eventos del DOM define 3 fases de propagación de eventos:

1. Fase de **captura** (*capture*): el evento desciende hasta el elemento
2. Fase de objetivo: el evento alcanza al elemento
3. Fase de **burbuja** (*bubbling*): el evento se propaga hacia arriba

La fase de captura:

* Con addEventListener, se pueden asociar handlers tanto a la fase de captura como de burbuja. Por defecto, se asocian a la fase de burbuja. Se podrían asociar a la fase de captura de este modo:

elem.addEventListener(..., {capture: true})

// o, solo "true" es una forma más corta de {capture: true}

elem.addEventListener(..., true)

* La fase de captura casi nunca se utiliza.
* Sólo se puede manejar con addEventListener (no se puede de los otros dos métodos)

Quitar un manejador: removeEventListener necesita la misma fase

* Si hemos asociado un manejador con addEventListener a una fase,
* para eliminarlo, hay que indicar con removeEventListener, hay que señalar esa fase:

addEventListener(..., true)

removeEventListener(..., true)

## event.target

**El elemento anidado más profundo que causó el evento es llamado elemento objetivo, accesible como event.target**

Cada manejador puede acceder a las propiedades del objeto event:

* event.target – el elemento más profundo que originó el evento.
* event.currentTarget (=this) – el elemento actual que maneja el evento (el que tiene al manejador en él)
* event.eventPhase – la fase actual (captura=1, objetivo=2, propagación=3).

## Detener la propagación

Cualquier manejador de evento puede detener el evento con event.stopPropagation()

No se recomienda utilizarlos, porque no podemos realmente asegurar que no lo necesitaremos más adelante, quizá para completar diferentes cosas.

# 3. Un patrón de manejo de eventos: delegación de eventos

La captura y la propagación nos permiten implementar un patrón de manejo de eventos llamado "delegación de eventos".

La idea es que si tenemos muchos elementos manejados de forma similar podemos:

* En lugar de asignar un manejador a cada uno de ellos, poner un único manejador a un ancestro común.
* En ese manejador utilizaremos event.target para ver dónde ocurrió realmente el evento.
* Si el manejador ocurrió en un evento que nos interesa, manejar el evento.

**Ejemplo 1: Diagrama Pa kua**

**Ejemplo 2: Un menú, con botones.** *(AVANZADO)*

* En el HTML, cada <button> tiene una propiedad data-action
* Los <button> están dentro de un <div>
* En JavaScript se crea un objeto asociado al <div> que ejecute el método de cada botón.

**Ejemplo 3: Comportamiento "contador"**

* En el HTML, se crea un atributo que agrega un comportamiento (ej. data-counter)
* En JavaScript, se asocia un manejador de eventos a document. Si el evento ocurre en un elemento (evento.target) que tenga ese atributo, se ejecuta la acción

**Ejemplo 4: Comportamiento "conmutador" (toggle)**

* En el HTML, se crea un atributo que agrega un comportamiento (ej. data-toggle-id)
* En JavaScript, se asocia un manejador de eventos a document. Si el evento ocurre en un elemento (evento.target) que tenga ese atributo, se ejecuta la acción, que consiste en ocultar/mostrar (toggle) el elemento con ese id

# 4. Acciones predeterminadas del navegador

Muchos eventos provocan determinadas acciones en el navegador.

Ejemplos:

* Al hacer clic sobre un <a href="…">, el navegador nos lleva a esa página.
* Al hacer clic en el botón de tipo “submit” de un formulario, la información del formulario se envía.
* Al pulsar con el ratón sobre un texto y mover el ratón, el texto se selecciona.

Si NO queremos que se ejecuten esas acciones predeterminadas, ¿qué tenemos que hacer?

Hay dos formas de hacerlo:

* Utilizando el objeto evento: con el método evento.preventDefault().

Ejemplos para los tres métodos de asignar eventos:

<a href="/" onclick="event.preventDefault()">aquí</a>

elemento.onclick = function(evento) {

...

evento.preventDefault();

...

}

elemento.addEventListener(function(evento) {

...

evento.preventDefault();

...

})

* Si el handler devuelve false. Esto sólo funciona si el handlser se asigna con on<evento> (no con addEventListener).

Ejemplos para los dos métodos de asignar eventos (con addEventListener no funciona).

<a href="/" onclick="return false">Haz clic aquí</a>

elemento.onclick = function(evento) {

...

return false;

}

# ~~5. Envío de eventos personalizados~~

Ampliación.