

{JS}

Clase 29

JS

Eventos personalizados

Eventos personalizados

Crear un evento personalizado en Javascript es muy sencillo. Se basa en crear una instancia del objeto CustomEvent, al cuál le pasaremos un con el nombre que le pondremos a nuestro evento. Como segundo parámetro le indicaremos un de opciones, que explicaremos más adelante.

```
const messageEvent = new CustomEvent("message", options);
```

Eventos personalizados

Nombre

En ejemplos sencillos no suele importar demasiado, pero una buena práctica a largo plazo es comenzar eligiendo una buena convención de nombres para los nombres de eventos, que sea «autoexplicativo» en cuanto la acción que vamos a realizar y a la vez sea coherente y fácil de recordar.

Eventos personalizados

Opciones

El segundo parámetro del CustomEvent es un `Object` donde podremos especificar varios detalles en relación al comportamiento o contenido del evento.

| Opciones | Valor inicial | Descripción |
|----------------------|-------------------------|--|
| <code>OBJECT</code> | <code>detail</code> | <code>null</code> Objeto que contiene la información que queremos transmitir. |
| <code>BOOLEAN</code> | <code>bubbles</code> | <code>false</code> Indica si el evento debe burbujear en el DOM «hacia la superficie» o no. |
| <code>BOOLEAN</code> | <code>composed</code> | <code>false</code> Indica si la propagación puede atravesar Shadow DOM o no. <u>Ver WebComponents</u> |
| <code>BOOLEAN</code> | <code>cancelable</code> | <code>false</code> Indica si el comportamiento se puede cancelar con <code>.preventDefault()</code> . |

Eventos personalizados

Opciones

El segundo parámetro del CustomEvent es un `Object` donde podremos especificar varios detalles en relación al comportamiento o contenido del evento.

| Opciones | Valor inicial | Descripción |
|----------------------|-------------------------|---|
| <code>OBJECT</code> | <code>detail</code> | null Objeto que contiene la información que queremos transmitir. |
| <code>BOOLEAN</code> | <code>bubbles</code> | <code>false</code> Indica si el evento debe burbujear en el DOM «hacia la superficie» o no. |
| <code>BOOLEAN</code> | <code>composed</code> | <code>false</code> Indica si la propagación puede atravesar Shadow DOM o no. <u>Ver WebComponents</u> |
| <code>BOOLEAN</code> | <code>cancelable</code> | <code>false</code> Indica si el comportamiento se puede cancelar con <code>.preventDefault()</code> . |

Eventos personalizados

```
const MessageEvent = new CustomEvent("user:data-message", {  
    detail: {  
        from: "Manz",  
        message: "Hello!"  
    },  
    bubbles: true,  
    composed: true  
});
```

Emisión de eventos

Eventos de formulario

Además de capturar eventos realizados por los usuarios, también podemos lanzarlos de forma programática

¿Para qué puede servir esto?

Con esto podemos simular eventos de usuarios o gestionar eventos personalizados

Emisión de eventos

```
<button>Click me</button>
<span class="text">Hover me</span>

<script>
const button = document.querySelector("button");
const text = document.querySelector(".text");

button.addEventListener("click", () => alert("Has pulsado el botón"));

text.addEventListener("mouseenter", () => {
  const event = new Event("click");
  button.dispatchEvent(event);
});
</script>
```

Emisión de eventos

```
<button id="botonRegistro">Registrar Usuario</button>
<div id="mensajeBienvenida" style="display:none;">¡Bienvenido al sistema!</div>

<script>
    // Paso 1: Crear el manejador de eventos para escuchar el evento personalizado
    document.addEventListener('usuarioRegistrado', function(evento) {
        console.log('Evento personalizado recibido:', evento);
        console.log('Detalles del evento:', evento.detail);

        document.getElementById('mensajeBienvenida').style.display = 'block';
    });

    // Paso 2: Crear y lanzar el evento personalizado al hacer clic en el botón
    document.getElementById('botonRegistro').addEventListener('click', function() {
        // Crear un evento personalizado con algunos detalles
        var eventoPersonalizado = new CustomEvent('usuarioRegistrado', {
            detail: {
                nombreUsuario: 'Juan',
                motivo: 'Registro completado'
            }
        });

        // Lanzar el evento
        document.dispatchEvent(eventoPersonalizado);
    });
</script>
```

Propagación de eventos

Propagación de eventos

El concepto de propagación de eventos es muy importante para entender todo el ecosistema de utilidades que rodean a la gestión de eventos.

La propagación hace referencia a que los eventos pueden gestionarse desde un elemento más profundo hasta la superficie, por eso se denomina comportamiento de burbuja.

Propagación de eventos

Dicho de otra forma, en una jerarquía de elementos, pueden capturarse eventos desde el nivel más profundo hasta el nivel más superficial, cuando se activa el evento más profundo.

Propagación de eventos

Veamos en qué orden se ejecutan los eventos click.
A esto le llamamos el comportamiento de burbuja

Propagación de eventos

¿cómo cambiamos este comportamiento?

Con el tercer parámetro de addEventListener

Probemos de nuevo

Propagación de eventos

¿Qué pasa cuando no queremos que el evento se propague y muera en el elemento en el que se ha producido?

stopPropagation al rescate

Ejemplo con el número de veces de ejecución por ejemplo

Propagación de eventos

Ejemplos

1. Prevenir la Propagación de un Evento de Clic en un Botón
Dentro de un Div
2. Detener la Propagación de un Evento de Menú
Contextual
3. Evitar que un Evento Keydown se Propague
4. Detener la Propagación de un Evento en un Manejador
de Eventos Delegado

Cancelación de eventos

1. Prevenir la Propagación de un Evento de Clic en un Botón Dentro de un Div

```
<div id="divPadre" style="padding: 20px; background-color: #f0f0f0;">  
    Haz clic en cualquier lugar de este div.  
    <button id="botonHijo">Haz clic en mi</button>  
</div>  
<script>  
    document.getElementById('divPadre').addEventListener('click', function() {  
        alert("Clic en el div padre!");  
    });  
  
    document.getElementById('botonHijo').addEventListener('click', function(evento) {  
        evento.stopPropagation();  
        alert("Clic en el botón hijo!");  
    });  
</script>
```

Cancelación de eventos

2. Detener la Propagación de un Evento de Menú Contextual

```
<div id="divExterno" style="padding: 20px; background-color: lightgray;">
    Div externo (clic derecho aqui mostrará el menú contextual predeterminado)
<div id="divInterno" style="padding: 20px; background-color: lightblue;">
    Div interno (clic derecho aqui no propagará el evento)
</div>
</div>
<script>
    document.getElementById('divInterno').addEventListener('contextmenu', function(evento) {
        evento.stopPropagation();
        alert("Menú contextual personalizado!");
        // Aqui podrias abrir tu propio menú contextual
        evento.preventDefault(); // Prevenir el menú contextual predeterminado del navegador
    });
</script>
```

Cancelación de eventos

3. Evitar que un Evento Keydown se Propague

```
<input type="text" id="miInput" placeholder="Escribe algo aqui...">
<script>
    document.getElementById('miInput').addEventListener('keydown', function(evento) {
        if(evento.key === "Enter") {
            evento.stopPropagation();
            alert("Presionaste Enter, pero el evento no se propagará.");
        }
    });
</script>
```

Cancelación de eventos

4. Detener la Propagación de un Evento en un Manejador de Eventos Delegado

```
<ul id="listaPadre">
  <li>Ítem 1</li>
  <li id="itemEspecial">Ítem Especial (clic detiene la propagación)</li>
  <li>Ítem 3</li>
</ul>
<script>
  document.getElementById('listaPadre').addEventListener('click', function() {
    alert("Clic en un ítem de la lista!");
  });

  document.getElementById('itemEspecial').addEventListener('click', function(evento) {
    evento.stopPropagation();
    alert("Clic en el ítem especial!");
  });
</script>
```

Cancelación de eventos

Cancelación de eventos

Para cancelar un evento se debe diferenciar entre dos escenarios: eventos predeterminados y eventos personalizados.

Cancelación de eventos

Con esto conseguiremos que no se ejecute el comportamiento por defecto de un evento como por ejemplo submit, los enlaces, etc...

1. Prevenir el Envío Automático de un Formulario
2. Prevenir que un Enlace Navegue a una URL
3. Prevenir el Menú Contextual del Botón Derecho del Ratón
4. Controlar el Comportamiento de Teclas Específicas

Cancelación de eventos

1. Prevenir el Envío Automático de un Formulario

```
<form id="miFormulario">
  <input type="text" name="nombre" required>
  <input type="submit" value="Enviar">
</form>
<script>
  document.getElementById('miFormulario').addEventListener('submit', function(evento) {
    evento.preventDefault(); // Previene el envio del formulario
    // Aqui tu lógica de validación
    if(/* validación exitosa */) {
      // Eventualmente, podrias enviar el formulario programáticamente
      // evento.target.submit();
    } else {
      alert("La validación falló.");
    }
  });
</script>
```

Cancelación de eventos

2. Prevenir que un Enlace Navegue a una URL

```
<a href="https://ejemplo.com" id="miEnlace">Haz clic en mi</a>
<script>
  document.getElementById('miEnlace').addEventListener('click', function(evento) {
    evento.preventDefault(); // Previene la navegación
    console.log("El enlace fue clickeado, pero no te llevará a ejemplo.com");
  });
</script>
```

Cancelación de eventos

3. Prevenir el Menú Contextual del Botón Derecho del Ratón

```
<div id="miElemento" style="width: 200px; height: 200px; background-color: lightblue;">  
  Haz clic derecho sobre mí  
</div>  

```

Cancelación de eventos

4. Controlar el Comportamiento de Teclas Específicas

```
<script>

  window.addEventListener('keydown', function(evento) {
    if(evento.code === "Space") {
      evento.preventDefault(); // Previene el scroll por la tecla Espacio
      console.log("Se presionó la tecla Espacio, pero no hará scroll en la página.");
    }
  });
</script>
```

Cancelación de eventos

5. Prevenir la Selección de Texto

```
<div id="noSelectable">Intenta seleccionarme</div>
<script>
  document.getElementById('noSelectable').addEventListener('mousedown', function(evento) {
    evento.preventDefault(); // Previene la selección de texto
  });
</script>
```

Cancelación de eventos

Con esto conseguiremos que no se ejecute el comportamiento por defecto de un evento como por ejemplo submit, los enlaces, etc...

Cancelación de eventos

Sin embargo, lo que se ha hecho es cancelar el comportamiento por defecto, no el evento en sí. Para anular completamente un evento se recurre a `removeEventListener()`. Además, hay que tener en cuenta que la anulación no puede hacerse cuando la definición del evento se hace con una función anónima, solo es posible cuando se utilizan funciones con identificador.

target vs currentTarget

target vs currentTarget

En el contexto de los eventos en JavaScript, target y currentTarget son propiedades del objeto evento que ofrecen información sobre el elemento que desencadenó el evento y el elemento que actualmente está manejando el evento, respectivamente. Aunque a primera vista puedan parecer similares, tienen propósitos distintos. Veamos las diferencias:

target vs currentTarget

event.target

- **Definición:** event.target se refiere al elemento que fue el objetivo original del evento. Es decir, el elemento en el que el evento se originó realmente. Si tienes un botón dentro de un div y haces clic en el botón, event.target sería el botón, independientemente de en qué elemento se haya registrado el manejador de eventos.
- **Uso Común:** Es especialmente útil en la delegación de eventos, donde un solo manejador de eventos en un elemento padre se utiliza para manejar eventos de múltiples elementos hijos. event.target te permite determinar cuál de los elementos hijos desencadenó el evento.

target vs currentTarget

event.currentTarget

- **Definición:** event.currentTarget se refiere al elemento en el que actualmente se está manejando el evento. A diferencia de event.target, que señala el origen del evento, event.currentTarget apunta al elemento que tiene el manejador de eventos que actualmente está procesando el evento.
- **Uso Común:** event.currentTarget es útil cuando el mismo manejador de eventos se ha adjuntado a múltiples elementos y necesitas saber cuál de ellos está procesando el evento actualmente. Esto es común en los casos donde se agrega el mismo manejador de eventos a varios elementos para evitar la duplicación de código.

target vs currentTarget

```
<div id="divPadre">
  <button id="botonHijo">Haz clic en mi</button>
</div>
<script>
  document.getElementById('divPadre').addEventListener('click', function(evento) {
    console.log('event.target:', evento.target); // Será el botón, si se clickea el botón
    console.log('event.currentTarget:', evento.currentTarget); // Siempre será divPadre
  });
</script>
```

<Despedida>

Email

bienvenidosaez@gmail.com

Instagram

@bienvenidosaez

Youtube

youtube.com/bienvenidosaez