Programador JavaScript Avanzado

Unidad 2: Eventos y Formularios

Contactos: tecnologia@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 2: Eventos y Formularios

Objetivos

Que el alumno logre:

- Conocer e implementar Eventos y Callbacks

Formularios

Los formularios y elementos, como <input>, tienen muchos eventos y propiedades especiales.

Los formularios son parte del objeto document.forms.

Podemos acceder a los formularios a través de un índice o el name de un elemento:

```
document.forms.miformu; // el formulario con name="miformu"
document.forms[0]; // el primer formulario en el documento
```

Cuando tenemos un formulario, cualquier elemento se encuentra disponible en form.elements.

```
<form name="miformu">
  <input name="uno" value="1">
  <input name="dos" value="2">
</form>

<script>
  // obtención del formulario
  let formulario = document.forms.miformu; // elemento <form name="
miformu">

  // get the element
  let elemento = formulario.elements.uno; // elemento <input
name="uno">

  alert(elemento.value); // 1
</script>
```

Podemos tener varios elementos con el mismo name, por ejemplo en input con type="radio" o checkbox

En ese caso form.elements[name] es un vector. Por ejemplo:

```
<form>
  <input type="radio" name="edad" value="10">
  <input type="radio" name="edad" value="20">
</form>

<script>
let formu = document.forms[0];
let edades = formu.elements.edad;
alert(edades[0]); // [object HTMLInputElement]
</script>
```

Estas propiedades de navegación no dependen de la estructura de las etiquetas. Todos los controles, sin importar qué tan profundos se encuentren en el formulario, están disponibles en form.elements.

Eventos: change, input, cut, copy, paste

Veamos varios eventos que acompañan la actualización de datos.

Evento change

El evento **change** se activa cuando el elemento finaliza un cambio.

Para ingreso de texto significa que el evento ocurre cuando se pierde foco en el elemento.

Por ejemplo, mientras estamos escribiendo en el siguiente cuadro de texto, no hay evento. Pero cuando movemos el focus (enfoque) a otro lado, por ejemplo hacemos click en un botón, entonces ocurre el evento change:

```
<input type="text" onchange="alert(this.value)">
<input type="button" value="Botón">
```

Para otros elementos: select, input type=checkbox/radio se dispara inmediatamente después de cambiar la opción seleccionada:

```
<select onchange="alert('Seleccionaste el día '+this.value)">
    <option value="">Selecciona un día</option>
    <option value="Lunes">Lunes</option>
    <option value="Miércoles">Miércoles</option>
    <option value="Viernes">Viernes</option>
</select>
```

Evento input

El evento input se dispara cada vez que un valor es modificado por el usuario.

A diferencia de los eventos de teclado, ocurre con el cambio a cualquier valor, incluso aquellos que no involucran acciones de teclado: copiar/pegar con el mouse o usar reconocimiento de voz para dictar texto.

Por ejemplo:

```
<input type="text" id="texto"> Evento oninput: <span
id="resultado"></span>
<script>
  texto.oninput = function() {
    resultado.innerHTML = texto.value;
```

```
  };
</script>
```

Para manejar cualquier modificación en un <input> este evento es la mejor opción.

Eventos: cut, copy, paste

Estos eventos ocurren al cortar/copiar/pegar un valor.

Estos pertenecen a la clase ClipboardEvent y dan acceso a los datos cortados/copiados/pegados.

También podemos usar event.preventDefault() para cancelar la acción y que nada sea cortado/copiado/pegado.

El siguiente código también evita todo evento cut/copy/paste y muestra qué es los que estamos intentando cortar/copiar/pegar:

```
<input type="text" id="campo">
<script>
    campo.onpaste = function(event) {
    alert("Está pegando lo siguiente: " +
event.clipboardData.getData('text/plain'));
    event.preventDefault();
  };

  campo.oncut = campo.oncopy = function(event) {
    alert(event.type + '-' + document.getSelection());
    event.preventDefault();
  };
</script>
```

En los manejadores cut y copy, llamamos a **event.clipboardData.getData(...)** que devuelve un string vacío. Esto es porque el dato no está en el portapapeles aún. Y si usamos **event.preventDefault()** no será copiado en absoluto.

En el ejemplo usamos document.getSelection() para obtener el texto seleccionado.

No solo es posible copiar/pegar texto, sino cualquier cosa. Por ejemplo, podemos copiar un archivo en el gestor de archivos del SO y pegarlo.

Esto es porque **clipboardData** implementa la interfaz **DataTransfer**, usada comúnmente para "arrastrar y soltar" y "copiar y pegar".

Aclaraciones

El portapapeles es algo a nivel "global" del Sistema Operativo, por lo tanto, un usuario puede alternar entre ventanas, copiar y pegar diferentes cosas, y el navegador no debería ver todo eso.

Por esto, la mayoría de los navegadores dan acceso al portapapeles únicamente bajo determinadas acciones del usuario, como copiar y pegar.

Evento y Método Submit

El evento **submit** se activa cuando el formulario es enviado, normalmente se utiliza para validar el formulario antes de ser enviado al servidor o bien para abortar el envío y procesarlo con JavaScript.

El método **form.submit()** permite iniciar el envío del formulario mediante JavaScript. Podemos utilizarlo para crear y enviar nuestros propios formularios al servidor.

Evento: submit

Mayormente un formulario puede enviarse de dos maneras:

- La **primera** – Haciendo click en **<input type="submit">**.
- La **segunda** – Pulsando la tecla **Enter** en un campo del formulario.

Ambas acciones causan que el evento **submit** sea activado en el formulario. El handler puede comprobar los datos, y si hay errores, mostrarlos e invocar ***event.preventDefault()***, entonces el formulario no será enviado al servidor.

```
<form onsubmit="alert('Enviado!'); return false">
    <p>Enter en el campo de texto <input type="text"
value="texto"></p>
    <p>Click en el botón "Enviar": <input type="submit"
value="Enviar"></p>
</form>
```

Ambas acciones muestran **alert('Enviado!')** y el formulario no es enviado debido a la presencia de return false.

Método Submit

Para enviar un formulario al servidor manualmente, podemos usar form.submit().

El evento submit no será generado. Se asume que si el programador llama form.submit(), entonces el script ya realizó todo el procesamiento relacionado.

A veces es usado para crear y enviar un formulario manualmente, por ejemplo:

```
let formulario = document.createElement('form');
formulario.action = 'https://google.com';
formulario.method = 'GET';

formulario.innerHTML = '<input name="google" value="test">';

// el formulario debe estar en el document para poder enviarlo
document.body.append(formulario);

formulario.submit();
```

Resumen

En esta Unidad…

Trabajamos con Eventos y formularios

En la próxima Unidad…

Trabajaremos con AJAX