UT5 Gestión de Eventos y Formularios en JavaScript

Desarrollo Web en entorno Cliente (DEW)

Profesora: Yaiza Guanche Pérez
yguaper@gobiernodecanarias.org

CIFP César Manrique
Centro Integrado de Formación Profesional

Tabla de Contenidos

Modelo de Gestión de Eventos

- El Evento es un mecanismo que se asocia cuando el usuario realiza una interacción o cambio sobre la página web.
- Capturar un evento es programar una acción para que se realice una tarea.
- El encargado de gestionar los eventos es el DOM (Document Object Model).

Modelo de Gestión de Eventos

Existen 4 modelos de eventos:

- Modelo de registro de eventos en línea.
- Modelo de registro de eventos tradicional.
- Modelo de eventos avanzados del W3C.
- Modelo de eventos de Microsoft.

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

Cada elemento XHTML tiene sus posibles eventos como propiedades: puede tener un evento de cada tipo. El nombre del evento es "on" seguido del nombre de la acción.

- Manejador como atributo de una etiqueta XHTML.

```
<h2 id="text" onclick="this.innerHTML='Eventos con Javascript'"
onmouseover="this.style.background='blue'"
onmouseout="this.style.background='white'">Texto con acción, pasa por encima (this)</h2>
```

¿Qué hace este código?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

Usar this es lo más adecuado en este caso, si no lo usamos accederíamos de la siguiente forma:

```
<h2 id="text" onclick="document.getElementById('text').innerHTML='Eventos con Javascript'"
onmouseover="document.getElementById('text').style.background='blue'"
onmouseout="document.getElementById('text').style.background='white'">Texto con acción,
pasa por encima (document.getElementById)</h2>
```

¿Qué hace este código?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

- Manejador como funciones externas:

```
<h2 onclick="cambiar_texto(this)">Pulsa aquí</h2>
<script>
    function cambiar_texto(elem) {
        elem.innerHTML = "Eventos con Javascript";
    }
</script>
```

¿Qué hace este código?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

Hay acciones que desencadenan varios eventos. Para evitar que el navegador ejecute la acción por defecto necesitamos añadir "return false;". Ejemplo:

```
<a href="http://www.google.es" onclick="alertar(); return false;">Pulsa aquí para ir a Google</a>
<script>
    function alertar() {
        alert ("Vamos a google");
    }
</script>
```

¿Qué hace este código?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

```
<a href="http://www.google.es" onclick="return preguntar();">Pulsa aquí para ir a Google</a>
<script>
    function preguntar() {
        return confirm ("¿Quieres ir a google?");
    }
</script>
```

¿Qué hace este código?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos en Línea.

- Carga de página con evento onload:

```
<body onload="alert('La página se ha cargado correctamente')">
```

O podemos indicarle una función a ejecutar una vez haya cargado, con function <nombre> (), no se ejecutará hasta que no se haya cargado todo el HTML.

NOTA: Este modelo de registro de eventos en línea es el más sencillo y menos recomendable, porque estamos incrustando el código JavaScript en el HTML.

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos Tradicional

- En este modelo se separa el código HTML del código JavaScript.

```
<h2 id="tradicional">Pulsa aquí (tradicional)</h2>
<script>
    document.getElementById("tradicional").onclick = cambiar; // sin paréntesis porque si no se ejecutaría la función.
    function cambiar() {
        alert("Entramos a cambiar");
        document.getElementById("tradicional").innerHTML="Modelo de registro de eventos tradicional";
        document.getElementById("tradicional").onclick = null;
    }
</script>
```

¿Inconveniente de este modelo?

Modelo de Gestión de Eventos.
Modelo de Registro de Eventos Tradicional

- En este modelo se separa el código HTML del código JavaScript.

```
<h2 id="tradicional2">Pulsa aquí (tradicional2)</h2>
<script>
    window.onload = function () {
        alert("La página se ha cargado correctamente");
        document.getElementById("tradicional2").onclick = miMensaje; //sin paréntesis para que no se ejecute
    }
    function miMensaje() {
        document.getElementById("tradicional2").innerHTML = "Modelo de registro de eventos tradicional2";
    }
</script>
```

Modelo de Gestión de Eventos. Modelo de Eventos Avanzados del W3C

Éste es el modelo correcto para crear los eventos y usar lo manejadores.

La especificación DOM define dos métodos denominados addEventListener () y removeEventListener () para asociar y desasociar manejadores de eventos.

Sintaxis:

addEventListener (nombre_evento, funcion, true|false);

nombre_evento: normalmente es el nombre del evento sin on.ej: onclick --> click

funcion: el nombre de la función a invocar sin paréntesis.

true|false: si es true el manejador emplea en la fase de *captura* | si es false el manejador se asocia a la fase de *bubbling*. .

Modelo de Gestión de Eventos. Modelo de Eventos Avanzados del W3C

```
<h3 id="w3c">Modelo del W3C</h3>
<h3 id="w3c_f_anonima">Modelo del W3C con funciones anónimas</h3>
<script>
    /* Sintaxis */
    /* elemento.addEventListener(<evento_sin_on>, <funcion>, <false|true>); */
    document.getElementById("w3c").addEventListener("click", saludar, false);
    document.getElementById("w3c").addEventListener("click", colorear, false);
    document.getElementById("w3c").addEventListener("mouseover", fondo, false);
    function saludar() {
        alert("Hola JavaScript");
        document.getElementById("w3c").removeEventListener("click", saludar); //solo lo hace una vez
    }function colorear() {
        document.getElementById("w3c").style.color = "red";
    }function fondo() {
        document.getElementById("w3c").style.background = "blue";
    }
    document.getElementById("w3c_f_anonima").addEventListener("click", function () {
        this.style.background = "orange";
    });
</script>
```

Modelo de Gestión de Eventos. Modelo de Eventos de Microsoft

- NOTA:Versiones IE8 y anteriores no soporta addEventListener y hay que usar attachEvent.

```
<h2 id="Microsoft">Modelo de Microsoft</h2>
<h2 id="Microsoft_anonima">Modelo de Microsoft anónima</h2>
<script>
    document.getElementById("Microsoft").attachEvent("onclick", saludar_m);
    document.getElementById("Microsoft").attachEvent("onclick", colorear_m);
    document.getElementById("Microsoft").attachEvent("onclick", fondo_m);
    function saludar_m() {
       alert("Hola JavaScript");
       document.getElementById("Microsoft").detachEvent("onclick", saludar_m); //solo lo hace una vez
    }function colorear_m() {
        document.getElementById("Microsoft").style.color = "red";
    }function fondo_m() {
        document.getElementById("Microsoft").style.background = "blue";
    }
    document.getElementById("Microsoft_anonima").attachEvent("onclick", function () {
        this.style.background = "orange";
    });

</script>
```

Manejadores de Eventos

- Con los manejadores de eventos podemos responder a las acciones de los usuarios en la web. Es una acción que se va a manejar, por ejemplo, el evento de hacer clic en la pantalla, su manejador es onclic.
- Dentro de los manejadores de eventos podemos colocar tantas instrucciones como necesitemos separadas por ;. Cuando son varias instrucciones lo habitual es crear una función y poner en el manejador la invocación a la función.

Manejadores de Eventos

- Cuando hacemos addEventListener, el DOM detecta el evento y lanza el manejador. Nos creamos una función para ello normalmente, pero no le pasamos parámetros porque el navegador crea un objeto e automaticamente que almacena el evento que se ha producido.
- Podemos encontrar por internet códigos como éste:

```
if (!e) e=window.event;  //si no se ha creado, lo creo.
```

Utilización de formularios desde código

Seleccionar el formulario conociendo el id (hay cuatro formas:

- let formulario = document.getElementById("miFormulario");
- let formulario2 = document.forms["miFormulario"];
- let formulario3 = document.getElementsByTagName("form")[0];
- let formulario4 = document.forms[0];

Acceso a los miembros de un formulario

Seleccionar elementos de un formulario:

- forms.elements[] devuelve un array con todos los inputs del formulario.
- getElementById("idElemento") devuelve un elemento con un id determinado.
- getElementsByTagName("etiqueta") devuelve un array con elementos de un tipo de etiqueta (inputs, select,…)
- getElementsByName ("nombre") devuelve un array con elementos que tienen el mismo nombre (ejemplo radioButton).

NOTA: getElement devuelve un elemento y getElements devuelve un array.

Modificación de apariencia y comportamiento

- e.prevenDefault() cancela el evento si este es cancelable, sin detener el resto del funcionamiento del evento. No detiene las siguientes llamadas al evento producidas en el DOM.
- e.stopPropagation() evita que se llame a la propagación del mismo evento, lo detiene.
- e.cancelable, propiedad para comprobar si el evento es cancelable o no.

Validación y envío

Existen tres tipos de validación:

- Básica: HTML + JavaScript para validar
- HTML5: haciendo uso de HTML5 las etiquetas ya tienen unos atributos específicos para validar.
- HTML5 + JavaScript: validación avanzada, podemos personalizar los mensajes de error.

Validación y envío

Validación formulario básico con HTML + JavaScript:

Antes eran todos los campos de tipo text y luego con JavaScript controlábamos la validación o restricciones.

Validación y envío

Validación formulario con HTML5:

Ahora con HTML5, especificamos los campos con los distintos tipos de inputs y ponemos las opciones o restricciones que nos permiten los atributos de los elementos.

Validación y envío

Validación formulario con HTML5:

En w3schools HTML inputs, podemos ver todos los tipos de inputs y las restricciones que ya existen en HTML5. (HTML Input Atributes)

maxlength. tamaño máximo.

pattern. se pueden establecer patrones a los tipos de inputs: text, date, search, url, tel, email, and password.

required. es requerido (sea distinto de null)

Validación y envío

Validación avanzada de formulario con HTML5 + JavaScript:

Modificamos los mensajes de error con JavaScript partiendo de la validación de HTML5.

- checkValidaty() devuelve true si un elemento input contiene un dato válido.
- validationMessage: muestra el mensaje por defecto de la validación.
- setCustomValidity() fijamos la propiedad validationMessage por defecto de un elemento input.

  Ver más métodos y propiedades de validación en w3schools Form Validation.
- NOTA: innerHTML: sobreescribe el valor que hay en el elemento id.

Validación y envío

La validación en el lado del cliente es una verificación inicial y una característica importante para garantizar una buena experiencia de usuario; mediante la detección de datos no válidos en el lado del cliente, el usuario puede corregirlos de inmediato. Si el servidor lo recibe y, a continuación, lo rechaza; se produce un retraso considerable en la comunicación entre el servidor y el cliente que insta al usuario a corregir sus datos.

La validación en el lado del cliente se realiza con HMTL5 + JavaScript y en el lado del servidor se realiza con PHP.

Expresiones Regulares

- Una expresión regular es un patrón (pattern) de caracteres.
- En JavaScript un objeto RegExp es un patrón con propiedades y métodos.
- Sintaxis: */pattern/modifier(s);*

Utilización de Cookies

- Las Cookies son datos almacenados en nuestro ordenador en pequeños archivos de texto.
- Recuerdan la información de un usuario aunque se cierre el navegador o se desconecte del servidor.
- Podemos guardar el nombre de un visitante de la página, el número de veces que ha visitado la web,…
- Se guardan en forma de pares nombre=valor ej: grupo=752NNS

Escritura y lectura de Cookies

Crear una Cookie:

La cookie está asociada al documento, document.cookie

Podemos poner varios pares de nombre=valor por ejemplo:

```
document.cookie = "grupo = 752NNS; expires = Thu, 30 Nov 2021 23:50:00 UTC; path=/;";
```

Y añadirle otro par posteriormente:

```
document.cookie = "modulo = DEW;";
```

Podemos indicarle al navegador a qué ruta pertenece la cookie por defecto es a la página actual, path=/

Si luego modificamos la cookie, tenemos que modificar todos los parámetros (path, expires,…) si no, lo interpreta como una cookie nueva.

Podemos ir a la configuración del navegador y ver que está la cookie que hemos creado.

Escritura y lectura de Cookies

Leer una cookie:

```
let miCookie = document.cookie;
alert(miCookie);
```

Cuando mostramos document.cookie nos va a mostrar todas las cookies que tenemos separadas por ";", seguidos como si fueran una cadena pero se trata como tal.

Escritura y lectura de Cookies

Modificar una cookie:

```
document.cookie = "modulo = DOR";
alert(document.cookie);
```

De esta forma lo modificamos ahora modulo DEW se ha reemplazado por DOR. Es más sencillo trabajar de esta forma, con cookies simples que con cookies con más parámetros, si no, tendríamos que especificar todos los valores del resto de parámetros.

Escritura y lectura de Cookies

Borrar una cookie:

Fijando la expiración con una fecha de expiración anterior a la que nos encontramos actualmente borramos la cookie.

Lo habitual es poner 01 Jun 1970 00:00:01 UTC; //1º fecha que se contempla en Date()

```
document.cookie = "grupo =; expires = Thu, 01 JAN 1970 00:00:01 UTC;";
alert(document.cookie);
document.cookie = "modulo =; expires = Thu, 01 JAN 1970 00:00:01 UTC;";
alert(document.cookie);
```

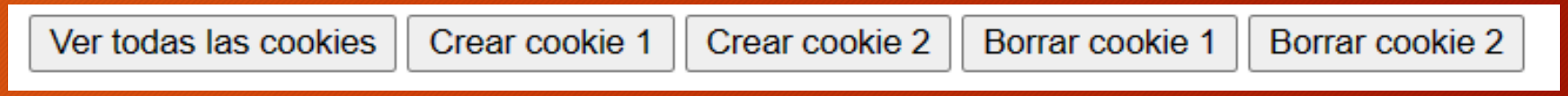
(Ver ejercicio UT5_Cookie_basico en evagd.)

Escritura y lectura de Cookies

Crear y Borrar una cookie con botones (eventos):

La forma más fácil pero no es la correcta, es que para cada una de los botones que tenemos, creamos una función distinta, entonces, si hay 10 cookies tendremos 10 funciones.

Eso no es eficiente, así que a pesar de tener 2 botones por cada cookie (crear y borrar), creamos funciones crearCookie, borrarCookie y verTodas asociadas a las acciones de los botones, e identificaremos mediante e.target.id el elemento que ha generado el evento.



(Ver ejercicio práctico UT5_Cookie_eventos en evagd.)

Escritura y lectura de Cookies

Crear, leer, modificar y borrar una cookie con botones (eventos) avanzado:

Un ejemplo más eficiente y escalable para trabajar con cookies es poner un botón genérico por cada acción permitida con las cookies, declarar las funciones genéricas de las cookies por acción(botón), y luego declarar unas funciones para pedirle al usuario la información de la cookie necesaria para operar con ella.

(Ver ejercicio práctico UT5_Cookie_avanzados en evagd.)