JavaScript Avanzado (Formularios y Eventos)

Contenido

1. Formularios

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. ***De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.***

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript

1.1. Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un ***array llamado forms*** y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que ***document. forms*** es el array que contiene todos los formularios de la página La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado ***elements*** por cada uno de los formularios de la página. Cada array ***elements*** contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que "el primer formulario de la página" ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array document.forms.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (*atributo name*) o a través de su atributo *id*.

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

- type: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (*text, button, checkbox, etc.*) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es *select-one*, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es *select-multiple*. Por último, en los elementos de tipo <textarea>, el valor de type es *textarea*.

- form: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar

```
document.getElementByld("id_del_elemento").form
```

- name: obtiene el valor del atributo name de HTML. Solamente se puede leer su valor, por lo que no se puede modificar.

- value: permite leer y modificar el valor del atributo value de HTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los elementos checkbox y radiobutton no es muy útil, como se verá más adelante.

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

- onclick: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir HTML (<input type="button">, <input type="submit">, <input type="image">).

- onchange: evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".

- onfocus: evento que se produce cuando el usuario selecciona un elemento del formulario.

- onblur: evento complementario de onfocus, ya que se produce cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

1.2. Utilidades básicas para formularios

Obtener el valor de los campos de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación, se muestra cómo obtener el valor de los campos de formulario más utilizados.

Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad value.

```
<input type="text" id="texto" />

var valor = document.getElementById("texto").value;

<textarea id="parrafo"x/textarea>

var valor = document.getElementById("parrafo").value;
```

Radiobutton

Cuando se dispone de un grupo de radiobuttons, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los radiobuttons se ha seleccionado. La propiedad ***checked*** devuelve *true* para el radiobutton seleccionado y false en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de radiobuttons-.

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI

<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO

<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

El siguiente código permite determinar si cada radiobutton ha sido seleccionado o no:

```
var elementos = document.getElementsByName("pregunta");

for(var i=0; icelementos.length; i++) {

   alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " +
elementos[i].checked);

}
```

Checkbox

Los elementos de tipo checkbox son muy similares a los radiobutton, salvo que en este caso se debe comprobar cada checkbox de forma independiente del resto. El motivo es que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los checkbox se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes checkbox:

```
<input type="checkbox" value="condiciones" name="condiciones" id="condiciones"/>
He leído y acepto las condiciones

<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/>

He leído la política de privacidad
```

Utilizando la propiedad checked, es posible comprobar si cada checkbox ha sido seleccionado:

```
var elemento = document.getElementByld("condiciones");

alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);

elemento = document.getElementById("privacidad");

alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

Select

Las listas desplegables (<select >) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

```
<select id="opciones" name="opcionesM>
   <option value="l">Primer valor</option>
   <option value="2">Segundo valor</option>
   <option value="3">Tercer valor</option>
   <option value="4">Cuarto valor</option>
</select>
```

En general, lo que se requiere es obtener el valor del atributo ***value*** de la opción (<option>) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

- ***options***, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante

```
document.getElementByld("id_de_la_lista").options[0].
```

- ***selectedlndex***, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array options creado automáticamente por el navegador para cada lista.

```
// Obtener La referencia a La Lista
var lista = document.getElementByld("opciones");
// Obtener el índice de La opción que se ha seLeccionado
var indiceSeleccionado = lista.selectedlndex;
// Con el índice y el array "options", obtener La opción seLeccionada
var opcionSeleccionada = lista.options[indiceSeleccionado];
// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;
alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " +
valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo ***value*** correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementByld("opciones");
// Obtener el valor de La opción seleccionada
var valorSeleccionado = lista.options[lista.selectedlndex].value;
// Obtener el texto que muestra la opción seleccionada
```

```
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad ***selectedIndex*** con el valor correspondiente a la propiedad ***value*** de la opción seleccionada.

Elemento <progress> (Indicador de Progreso)

El elemento <progress> se utiliza para mostrar el progreso de una tarea.

Atributo	Descripción	Tipo de Valor
value	La cantidad de trabajo completado.	Número (entre 0 y max)
max	La cantidad total de trabajo que se necesita.	Número

Puedes obtener y establecer sus valores directamente como propiedades del DOM.

```
// 1. Obtener el elemento
const progressBar = document.getElementById('miProgreso');

// 2. Leer el valor actual
console.log('Progreso actual:', progressBar.value);

// 3. Establecer un nuevo valor
progressBar.value = 65;

// 4. Cambiar el valor máximo
progressBar.max = 200;
```

Elemento <meter> (Medidor Escalar)

El elemento <meter> se usa para representar un valor en un rango conocido, como el uso de disco o la relevancia de un resultado.

Atributo	Descripción	Tipo de Valor
value	**El valor actual a medir.**	**Número**
min	**El valor mínimo del rango (por defecto 0).**	**Número**
max	**El valor máximo del rango (por defecto 1).**	**Número**
low	**Valor por debajo del cual se considera "bajo".**	**Número**
high	**Valor por encima del cual se considera "alto".**	**Número**
optimum	**El valor considerado óptimo o ideal.**	**Número**

```
// 1. Obtener el elemento
const meter = document.getElementById('miMedidor');

// 2. Establecer el uso de RAM al 85%
meter.value = 85;

// 3. Cambiar los umbrales de advertencia (high) y óptimo
meter.high = 90; // Advertencia a partir del 90%
meter.optimum = 60; // 60% es el mejor uso
```

Componente range (Selector de Rango)

Este es un control deslizante que permite al usuario seleccionar un valor dentro de un rango predefinido.

Atributo	Descripción	Tipo de Valor
value	El valor seleccionado actualmente.	Número
min	El valor mínimo que se puede seleccionar.	Número
max	El valor máximo que se puede seleccionar.	Número
step	El incremento/decremento permitido.	Número

```
// 1. Obtener el range y el elemento para mostrar el valor
const rangeSlider = document.getElementById('controlVolumen');
const volumen = document.getElementById('nivelVolumenActual');
// 2. Escuchar el evento 'input' o 'change' usualmente
rangeSlider.oninput = function () => {
  // Actualizar el texto mostrado
  volumen.textContent = = this.value;

  console.log('Volumen seleccionado:', volumen);
});

// 3. Establecer un valor inicial por JS
rangeSlider.value = 50;
volumen.textContent = rangeSlider.value;
```

Establecer el foco en un elemento

En programación, cuando un elemento está seleccionado y se puede escribir directamente en él o se puede modificar alguna de sus propiedades, se dice que tiene el foco del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla TABULADOR sobre una página web, los diferentes elementos van obteniendo el foco del navegador.

Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de HTML, se utiliza la función ***focus()***. El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo id es igual a primero:

```
if(document.forms.length > 0) {

   if(document.forms[0].elements.length > 0) {
      document.forms[0].elements[0].focus();

   }

<form id="formulario" action="#">

   <input type="text" id="primero" />

</form>
```

Ejercicio: Amplia el ejemplo anterior para que el foco se establezca en el primer campo del formulario que no esté oculto (*hidden*).

Ejercico: Si la conexión es muy lenta, el usuario puede dar dos veces al botón o link que envía la información del formulario al servidor. Para evitar esto, deshabilita el botón de envío después de la primera pulsación.

2. Eventos

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (if, for, while) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.
Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes eventos que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de programación basada en eventos.

En este tipo de programación, los scripts se dedican a esperar a que el usuario "haga algo" (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La

pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan ***"event handlers"*** en inglés y suelen traducirse por ***"manejadores de eventos"***.

2.1. Modelo básico de eventos

En este modelo, cada elemento o etiqueta HTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos HTML diferentes y un mismo elemento HTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo "***on***", seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina *onclick* y el evento asociado a la acción de mover el ratón se denomina *onmousemove*.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>

onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo, al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la "acción por defecto" de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo <input type="submit" > se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva.

Casi todos los eventos disponibles

Es imposible incluir absolutamente **TODOS** los eventos posibles en una sola tabla (algunos son muy específicos de APIs modernas o están obsoletos), pero aquí tienes unas tablas que abarcan la gran mayoría de los eventos estándar del DOM y la interfaz de usuario, categorizados para que sea más fácil de consultar:

Eventos de Ratón (Mouse Events)

Estos eventos se activan por las acciones del usuario con un dispositivo apuntador (como un ratón o *trackpad*).

Evento	Descripción	Elementos Comunes
click	Se hace clic en el elemento (presionar y soltar el botón).	Todos
dblclick	Se hace doble clic en el elemento.	Todos
mousedown	El botón del ratón es **presionado**.	Todos
mouseup	El botón del ratón es **liberado**.	Todos
mousemove	El ratón se mueve sobre el elemento.	Todos
mouseover	El cursor entra en el área del elemento (no burbujea).	Todos
mouseenter	El cursor entra en el área del elemento (no se dispara en descendientes).	Todos
mouseout	El cursor sale del área del elemento (no burbujea).	Todos
mouseleave	El cursor sale del área del elemento (no se dispara en descendientes).	Todos
contextmenu	El usuario intenta abrir el menú contextual (**normalmente clic derecho**).	Todos

Eventos de Formulario (Form Events)

Relacionados con la interacción con controles de formularios y la validación de datos.

Evento	Descripción	Elementos Comunes
submit	Un formulario es enviado.	<form>
reset	Un formulario es reiniciado.	<form>
change	El valor de un elemento cambia y pierde el foco.	<input>, <select>, <textarea>
input	El valor de un elemento cambia instantáneamente (mientras se escribe/arrastra).	<input>, <textarea>

Eventos de Foco y Teclado (Focus and Keyboard Events)

Relacionados con la entrada de texto y el estado de foco de un elemento.

Evento	Descripción	Elementos Comunes
focus	Un elemento recibe el foco.	<input>, <button>, <a>, etc.
blur	Un elemento pierde el foco.	<input>, <button>, <a>, etc.
focusin	Un elemento o uno de sus descendientes recibe el foco (burbujea).	Todos
focusout	Un elemento o uno de sus descendientes pierde el foco (burbujea).	Todos
keydown	Se presiona cualquier tecla.	document, <input>, <textarea>
keyup	Se libera cualquier tecla.	document, <input>, <textarea>
keypress	Se presiona una tecla que produce un carácter (obsoleto).	document, <input>, <textarea>

Eventos de Ventana y Documento (Window and Document Events)

Afectan al documento completo o a la ventana del navegador.

Evento	Descripción	Elementos Comunes
load	Se ha cargado completamente la página o un recurso (<img>).	window, document, recursos multimedia
unload	El usuario abandona la página (al cerrar o navegar).	window
beforeunload	Se dispara justo antes de que se descargue el recurso.	window
resize	El tamaño de la ventana (viewport) ha cambiado.	window
scroll	El usuario ha desplazado el contenido del elemento.	window, Elementos con *scroll*

error	Un recurso no pudo ser cargado (imagen, script, etc.).	window, <img>, <script>
hashchange	La parte *hash* (#fragmento) de la URL ha cambiado.	window
online	El navegador está conectado a la red.	window
offline	El navegador está desconectado de la red.	window
DOMContentLoaded	El documento HTML ha sido cargado y parseado (sin esperar CSS/imágenes).	document

Eventos Táctiles (Touch Events)

Específicos para dispositivos con pantalla táctil.

Evento	Descripción	Elementos Comunes
touchstart	Uno o más puntos de contacto son colocados sobre la superficie.	Todos
touchend	Uno o más puntos de contacto son removidos de la superficie.	Todos
touchmove	Uno o más puntos de contacto se mueven sobre la superficie.	Todos
touchcancel	Un evento de toque ha sido interrumpido (ej. demasiados puntos).	Todos

Eventos de Multimedia (Media Events)

Utilizados principalmente con los elementos <audio> y <video>.

Evento	Descripción	Elementos Comunes
play	El medio comienza o reanuda la reproducción.	<audio>, <video>
pause	La reproducción es pausada.	<audio>, <video>
ended	El medio ha finalizado la reproducción.	<audio>, <video>
volumechange	El volumen o el estado de silencio ha cambiado.	<audio>, <video>
progress	El navegador está descargando el medio.	<audio>, <video>
timeupdate	El tiempo de reproducción actual ha cambiado (frecuentemente).	<audio>, <video>
canplay	El medio puede comenzar a reproducirse (con almacenamiento en búfer).	<audio>, <video>

Eventos de Arrastrar y Soltar (Drag & Drop Events)

Permiten mover elementos usando el ratón. Requieren el atributo draggable="true".

Evento	Descripción	Elementos Comunes
dragstart	El usuario comienza a arrastrar un elemento.	Elementos con draggable="true"
drag	Un elemento se arrastra (dispara constantemente).	Elementos con draggable="true"
dragend	El arrastre ha finalizado (soltado o cancelado).	Elementos con draggable="true"
dragenter	El elemento arrastrado entra en una zona de destino.	Zonas de destino
dragleave	El elemento arrastrado sale de una zona de destino.	Zonas de destino
dragover	El elemento arrastrado está sobre la zona de destino (dispara constantemente).	Zonas de destino
drop	El elemento arrastrado es soltado en un destino.	Zonas de destino

Ejemplo:

```
<!DOCTYPE HTML>
<html>
<head>
    <style>
    #div1 {
    width: 350px;
    height: 70px;
    padding: 10px;
    border: 1px solid #aaaaaa;
    }
    </style>
    <script>
        document.getElementById("drag1").ondragstart = drag;

        document.getElementById("div1").ondrop = drop;
        document.getElementById("div1").ondragover = allowDrop;

        function allowDrop(ev) { // Permite que al pulsar sobre el texto no se
seleccione
            ev.preventDefault();
        }

        function drag(ev) {
            // dataTransfer guarda el dato pasado como segundo parámetro en la con el
nombre pasado como primer parámetro (Text). ev.target es el párrafo.
            ev.dataTransfer.setData("Text", ev.target.id);
        }

        function drop(ev) {
            ev.preventDefault();
            let data = ev.dataTransfer.getData("Text");
```

```
            ev.target.appendChild(document.getElementById(data));

        }
    </script>
</head>
<body>

    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <br>
    <p id="drag1" draggable="true" ondragstart="drag(event)">Arrastra el párrafo al
rectángulo.</p>

</body>
</html>
```

Eventos de Animación y Transición CSS

Se activan al cambiar propiedades CSS de forma animada.

Evento	Descripción	Elementos Comunes
transitionstart	Una transición CSS ha comenzado.	Todos
transitionend	Una transición CSS ha finalizado.	Todos
animationstart	Una animación CSS ha comenzado.	Todos
animationend	Una animación CSS ha finalizado.	Todos
animationiteration	Una iteración de la animación ha finalizado.	Todos

Eventos de Portapapeles (Clipboard Events)

Relacionados con las operaciones de copiar, cortar y pegar.

Evento	Descripción	Elementos Comunes
copy	El usuario intenta copiar contenido.	Elementos seleccionables
cut	El usuario intenta cortar contenido.	Elementos seleccionables
paste	El usuario intenta pegar contenido.	Elementos editables (<input>, con contenteditable)

2.2. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su

ejecución. Las funciones o código JavaScript que se definen para cada evento se denominan ***"manejador de eventos"*** y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos HTML.
- Manejadores como funciones JavaScript externas.
- Manejadores "semánticos".

Manejadores de eventos como atributos HTML

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento HTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" 1>
```

En este método, se definen atributos HTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es onclick. Así, el elemento HTML para el que se quiere definir este evento, debe incluir un atributo llamado onclick.

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas de pasar el ratón por encima');">
```

Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima .

Manejadores de eventos y variable this

JavaScript define una variable especial llamada **this** que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable *this* para referirse al elemento HTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del , el color del borde se muestra de color negro. Cuando el ratón sale del , se vuelve a mostrar el borde con el color gris claro original.

Elemento original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"> Sección de contenidos...</div>
```

Si no se utiliza la variable this, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor=silver';">

Sección de contenidos...

</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable this, que hace referencia al elemento HTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="this.style.borderColor=black;"
onmouseout="this.style.borderColor=silver';">
Sección de contenidos...
</div>
```

La definición de los manejadores de eventos en los atributos HTML es el método más sencillo, pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento HTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" 1>
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
   alert('Gracias por pinchar');
}
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento HTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable this y, por tanto, es necesario pasar esta variable como parámetro a la función:

```
<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">
Sección de contenidos...
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento.

Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos HTML y como funciones externas) tienen un grave inconveniente: **"ensucian" el código HTML de la página**.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (HTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (HTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos HTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos HTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
// Función externa
function muestraMensaje() {
   alert('Gracias por pinchar');
}
// Asignar La función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje; // ELemento HTML
<input id="pinchable" type="button" value="Pinchame y verás" />
```

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento HTML mediante el atributo id.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

S**e asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:**

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad onclick de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;
// Ejecutar una función y guardar su resultado en una propiedad de un elemento
```

```
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código HTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable **this** para referirse al elemento que provoca el evento.

addEventListener y removeEventListener

Existe otra manera de asignar eventos a un elemento HTML desde JavaScript. Consiste en usar el método ***document.addEventListener***. Este método acepta tres parámetros:

- el nombre del evento a controlar.
- el método que se ejecutará cuando este evento ocurra.
- (opcional) un valor booleano que especifica si el evento debe ejecutarse en la fase de captura (true) o en la fase de propagación (false).

```
document.addEventListener("click", function(){
  document.body.style.backgroundColor = "red";
});
```

Podemos asignar varios eventos al mismo elemento. Además, podemos asignar varias funciones para el mismo evento de un elemento. Ejemplo:

```
document.addEventListener("click", myFunction);
document.addEventListener("click", someOtherFunction);
```

Puede que nos sea útil que, llegado un momento, un determinado evento que hemos asignado con **addEventListener,** deje de ocurrir. Para eliminar eventos de un componente utilizaremos el método ***document.removeEventListener***. Los parámetros son: el evento a suprimir y la función que ejecuta ese evento.

```
document.addEventListener("mousemove", myFunction);

// ….
document.removeEventListener("mousemove", myFunction);
```

Hay que tener en cuenta que si asignamos un evento con una función anónima no podremos eliminar ese evento.

Paso de parámetros a la función que ejecuta el evento

Hay veces que necesitaremos ejecutar el evento con algún parámetro. Para poder hacer esto tendremos que usar funciones anónimas y llamar a otra función dentro de la que ejecuta el evento. Ejemplo:

```
var numClick = 0;

function executeClick(n) {

    n++;
```

```
        console.log(“Número de veces que has pulsado este elemento: “ + n);
        numClick = n;
}

// Función que ejecuta el evento:
elem.onclick = function() {
        executeClick(numClick);
}
```

Prevenir la carga inicial del evento cuando se crea (preventDefault())

Cancela el evento si este es cancelable, sin detener el resto del funcionamiento del evento, es decir, puede ser llamado de nuevo. Ejemplo:

```
document.getElementById("myAnchor").addEventListener("click", function(event){
  event.preventDefault()
……
});
```

Este **event.preventDefault()** hará que el evento no se ejecute cuando se crea, sino en las sucesivas veces que se haga click. Podemos hacerlo igualmente para el resto de eventos.

Otras soluciones son:

- Hacer que la función que ejecuta el evento devuelva **false** (return false).
- Usar el método **event.stopPropagation()** → detiene la propagación de un evento con el objetivo de que no se realice otra ejecución u otro listener lo escuche a través del DOM.

2.3. Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función, por ejemplo, se encarga de procesar el evento onclick, quizás necesite saber en qué posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante **conocer la tecla que se ha pulsado**, por ejemplo para diferenciar las teclas normales de las teclas especiales (ENTER, tabulador, Alt, ctrl ., etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado ***event***. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos crean mágicamente y de forma automática un argumento que se pasa a la función manejadora, por lo que no es

necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este "argumento mágico", sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

El objeto *event* se obtiene mágicamente a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(event) { … }
```

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

Información sobre el evento

La propiedad ***type*** indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad **type** devuelve el tipo de evento producido, que es igual al nombre del evento, pero sin el prefijo “on”.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

```
function resalta(e) {
  switch(e.type) {
   case 'mouseover':
      this.style.borderColor = 'black'; break;
   case 'mouseout':
      this.style.borderColor = 'silver'; break;
   }
}
window.onload = function() {
       document.getElementById("seccion").onmouseover = resalta;
       document.getElementById("seccion").onmouseout = resalta;
}
<div id="seccion" style="width:150px; height:60px; border:thin solid silver">
 Sección de contenidos...
</div>
```

Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y, por tanto, los más difíciles de manejar. En primer lugar, existen

muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (***onkeypress, onkeydown y onkeyup***). Por último, existen dos tipos de teclas: las **teclas normales** (como letras, números y símbolos normales) y las **teclas especiales** (como ENTER, Alt, Shift, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: onkeydown, onkeypress y onkeyup. El evento onkeydown se corresponde con el hecho de pulsar una tecla y no soltarla; el evento onkeypress es la propia pulsación de la tecla y el evento onkeyup hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento *onkeypress*. La información que proporcionan los eventos onkeydown y onkeyup se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función **String.fromCharCode()**. A continuación, se incluye un script que muestra toda la información sobre los tres eventos de teclado:

```
window.onload = function() {
   document.onkeyup = muestraInformacion;
   document.onkeypress = muestraInformacion;
   document.onkeydown = muestraInformacion;
}
function muestraInformacion(evento) {
   var mensaje = "Tipo de evento: " + evento.type +  "<br>" +
   "Código ASCCI: " +     evento.code + "<br>" +
   "Tecla pulsada: " + evento.key + "<br>" +
   info.innerHTML += "<br>" + mensaje
}
...
<div id="info"></div>
```

Podemos usar **e.code** y **e.key**. **e.code** devuelve el código ASCII de la tecla pulsada y **e.key** la propia tecla pulsada. Las teclas especiales no devuelven nada para e.key.

El código anterior puede variar entre distintos navegadores. Según el navegador y la tecla pulsada se pueden devolver distintos valores o ninguno. Cada navegador puede devolver el código interno, el código ASCII de la tecla pulsada o ningún valor. ***PROBAR EL CÓDIGO ANTERIOR EN DISTINTOS NAVEGADORES.***

Por último, las propiedades ***altKey, ctrlKey y shiftKey*** almacenan un valor booleano que indica si alguna de esas teclas estaba pulsada al producirse el evento del teclado. Sorprendentemente, **estas tres propiedades funcionan de la misma forma en todos los navegadores**.

Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las **coordenadas de la posición del puntero del ratón**. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, **el punto que se toma como referencia de las coordenadas puede variar**.

De esta forma, es posible obtener la posición del ratón respecto de la **pantalla del ordenador**, respecto de la **ventana del navegador** y respecto de **la propia página HTML**. Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades ***clientX*** y ***clientY***:

```
function muestralnformacion(evento) {
   var coordenadaX = evento.clientX;
   var coordenadaY = evento.clientY;
   alert(Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY);
}
document.onclick = muestralnformacion;
```

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades ***screenX*** y ***screenY***:

```
var coordenadaX = evento.screenX; var coordenadaY = evento.screenY;
```

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer scroll sobre la página web.

Las coordenadas respecto del origen de la página se obtienen mediante las propiedades ***pageX*** y ***pageY***.

Ejercicio: Crear un script que informe al usuario en que zona de la pantalla ha pulsado el ratón. Las zonas definidas son las siguientes: izquierda arriba, izquierda abajo, derecha arriba y derecha abajo.

Ejercicio: Limita el número máximo de caracteres que puede contener un textArea a *N* caracteres.

Ayuda: onkeypress, return false/true;

Anexo I: Listado de eventos

Eventos Estándar

Estos eventos se definen en las especificaciones Web oficiales, y deben ser comunes en todos los navegadores. Cada evento aparece junto con la interfaz que representa el objeto enviado a los destinatarios del evento (por

lo que puede encontrar información sobre los datos que se incluyen en cada caso), así como un enlace a la especificación o especificaciones que definen el evento.

Nombre del evento	Se activa al...
abort	La carga de un recurso ha sido abortada.
abort	La progresión se ha terminado (no debido a un error).
abort	Una transacción ha sido abortada.
afterprint	El documento asociado ha iniciado la impresión o la vista previa de impresión de ha cerrado
animationend	Una animation de CSS ha terminado.
animationiteration	Una animation de CSS se repite.
animationstart	Una animation de CSS ha iniciado.
audioprocess	El búfer de entrada de un ScriptProcessorNode está listo para ser procesado.
beforeprint	El documento asociado está a punto de ser impreso o previsualizado para imprimir.
beforeunload	
beginEvent	Inicia un elemento de animación SMIL.
blocked	Una conexión abierta en la base de datos está bloqueando una transacción versionchange en la misma base de datos.
blur	Un elemento ha perdido el foco (does not bubble).
cached	Los recursosn que aparecen en el manifiesto se han descargado, y la aplicación está ahora almacenada en caché.
canplay	La aplicación del usuario puede reproducir los medios, pero se estima que no hay datos suficientes, han sido cargados para reproducir el medio hasta el final sin tener que parar para una mayor amortiguación del contenido.
canplaythrough	La aplicación de usuario puede reproducir los medios, y se estima que hay datos suficientes, han sido cargados para reproducir el medio hasta el final sin tener que parar para una mayor amortiguación del contenido
change	Un elemento perdió el foco y su valor cambio desde que obtuvo el foco.
chargingchange	La batería inició o dejo de cargar
chargingtimechange	El atributo chargingTime se ha actualizado.
checking	La aplicación de usuario está comprobando una actualización o intenta descargar el manifiesto de caché por primera vez.
click	Un botón en el dispositivo señalador se ha pulsado y soltado en un elemento.
close	Una conexión WeSocket se ha cerrado.
compassneedscalibration	La brujula usada para obtener los datos de la orientación en que está necesita calibración.
complete	
complete	El renderizado de un OfflineAudioContext finaliza.
compositionend	La composición de un pasaje de texto se ha completado o cancelado.
compositionstart	La composición de un pasaje de un texto está preparado (similar a KeyDown para una entrada de teclado, sino que trabaja con otros insumos, como el reconocimiento de voz).
compositionupdate	Un carácter se añade a un pasaje de texto que está siendo compuesto.
contextmenu	Se hace clic en el botón derecho del ratón (antes de que aparezca el menú contextual).
copy	El texto seleccionado se ha agregado al porta papeles.
cut	El texto seleccionado ha sido borrado del documento y agregado al portapapeles.
dblclick	Un botón del dispositivo señalador hace click dos veces en un elemento.

devicelight	Datos nuevos están disponibles desde un sensor de luz.
devicemotion	Datos nuevos están disponibles desde un sesor de movimiento.
deviceorientation	Datos nuevos están disponibles desde un sensor de orientación.
deviceproximity	Datos nuevos están disponibles desde un sensor de proximidad (indica una distancia aproximada entre el dispositivo y un objeto cercano).
dischargingtimechange	El atributo dischargingTime se ha actualizado.
DOMActivate Obsoleto	Un botón, enlace o elemento de estado cambiante esta activado (usa click en su lugar).
DOMAttributeNameChanged Obsoleto	El nombre de un atributo cambiado (usa mutation observers en su lugar).
DOMAttrModified Obsoleto	El valor de un atributo ha sido cambiado (usa mutation observers en su lugar).
DOMCharacterDataModified Obsoleto	Un texto u otro CharacterData ha sido cambiado (usa mutation observers en su lugar).
DOMContentLoaded	El documento ha terminado de cargar (pero no sus recursos dependientes).
DOMElementNameChanged Obsoleto	El nombre de un elemento dependiente (usa mutation observers en su lugar).
DOMFocusIn Obsoleto	Un elemento ha recibido el foco (usa focus o focusin en su lugar).
DOMFocusOut Obsoleto	Un elemento ha perdido el foco (usa blur o focusout en su lugar).
DOMNodeInserted Obsoleto	Un nodo ha sido añadido como un hijo de otro nodo (usa mutation observers en su lugar).
DOMNodeInsertedIntoDocument Obsoleto	Un nodo ha sido insertado en el documento (usa mutation observers en su lugar).
DOMNodeRemoved Obsoleto	Un nodo ha sido eliminado de su nodo padre A node has been removed from its parent node (usa mutation observers en su lugar).
DOMNodeRemovedFromDocument Obsoleto	Un nodo ha sido eliminado del documento (usa mutation observers en su lugar).
DOMSubtreeModified Obsoleto	Ocurrio un cambio en el documento (usa mutation observers en su lugar).
downloading	La aplicación de usuario ha encontrado una actualización y está buscando o descargando los recursos enumerados en el caché del manifiesto por primera vez.
drag	Un elemento o texto seleccionato está siendo arrastrado (cada 35ms).
dragend	Una operación de arrastre ha finalizado (al pulsar un botón del ratón o pulsando la tecla escape).
dragenter	Un elemento arrastrado o un texto seleccionado entró en un destino válido.
dragleave	Un elemento arrastrado o un texto seleccionado salió de un destino válido.
dragover	Un elemento o texto seleccionado es arrastrado encima de un destino válido (cada 50ms).
dragstart	El usuario empezó a arrastrar un elemento o un texto seleccionado.
drop	Un elemento es soltado en un destino válido.
durationchange	El atributo duration se ha actualizado.
emptied	Los medios de comunicación se ha convertido en vacío, por ejemplo, este evento se envía si los medios de comunicación ya se ha cargado (o parcialmente cargado), y el método load() es llamado para volver a cargarlo.
ended	La reproducción se ha detenido porque se ha alcanzado el final de los medios de comunicación.
ended	
endEvent	Un elento de animación SMIL termina.
error	Un recurso no se pudo cargar.
error	La progresión a fallado.

error	Se ha producido un error al descargar el manifiesto de caché o actualizar el contenido de la aplicación.
error	A WebSocket connection has been closed with prejudice (some data couldn't be sent for example).
error	An event source connection has been failed.
error	A request caused an error and failed.
focus	An element has received focus (does not bubble).
focusin	An element is about to receive focus (bubbles).
focusout	An element is about to lose focus (bubbles).
fullscreenchange	An element was turned to fullscreen mode or back to normal mode.
fullscreenerror	It was impossible to switch to fullscreen mode for technical reasons or because the permission was denied.
gamepadconnected	A gamepad has been connected.
gamepaddisconnected	A gamepad has been disconnected.
hashchange	The fragment identifier of the URL has changed (the part of the URL after the #).
input	The value of an element changes or the content of an element with the attribute contenteditable is modified.
invalid	A submittable element has been checked and doesn't satisfy its constraints.
keydown	A key is pressed down.
keypress	A key is pressed down and that key normally produces a character value (use input instead).
keyup	A key is released.
levelchange	The level attribute has been updated.
load	A resource and its dependent resources have finished loading.
load	Progression has been successful.
loadeddata	The first frame of the media has finished loading.
loadedmetadata	The metadata has been loaded.
loadend	Progress has stopped (after "error", "abort" or "load" have been dispatched).
loadstart	Progress has begun.
message	A message is received through a WebSocket.
message	A message is received from a Web Worker.
message	A message is received from a child (i)frame or a parent window.
message	A message is received through an event source.
mousedown	A pointing device button (usually a mouse) is pressed on an element.
mouseenter	A pointing device is moved onto the element that has the listener attached.
mouseleave	A pointing device is moved off the element that has the listener attached.
mousemove	A pointing device is moved over an element.
mouseout	A pointing device is moved off the element that has the listener attached or off one of its children.
mouseover	A pointing device is moved onto the element that has the listener attached or onto one of its children.
mouseup	A pointing device button is released over an element.
noupdate	The manifest hadn't changed.
obsolete	The manifest was found to have become a 404 or 410 page, so the application cache is being deleted.
offline	The browser has lost access to the network.

online	The browser has gained access to the network (but particular websites might be unreachable).
open	A WebSocket connection has been established.
open	An event source connection has been established.
orientationchange	The orientation of the device (portrait/landscape) has changed
pagehide	A session history entry is being traversed from.
pageshow	A session history entry is being traversed to.
paste	Data has been transfered from the system clipboard to the document.
pause	Playback has been paused.
pointerlockchange	The pointer was locked or released.
pointerlockerror	It was impossible to lock the pointer for technical reasons or because the permission was denied.
play	Playback has begun.
playing	Playback is ready to start after having been paused or delayed due to lack of data.
popstate	A session history entry is being navigated to (in certain cases).
progress	In progress.
progress	The user agent is downloading resources listed by the manifest.
ratechange	The playback rate has changed.
readystatechange	The readyState attribute of a document has changed.
repeatEvent	A SMIL animation element is repeated.
reset	A form is reset.
resize	The document view has been resized.
scroll	The document view or an element has been scrolled.
seeked	A *seek* operation completed.
seeking	A *seek* operation began.
select	Some text is being selected.
show	A contextmenu event was fired on/bubbled to an element that has a contextmenu attribute
stalled	The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
storage	A storage area (localStorage or sessionStorage) has changed.
submit	A form is submitted.
success	A request successfully completed.
suspend	Media data loading has been suspended.
SVGAbort	Page loading has been stopped before the SVG was loaded.
SVGError	An error has occurred before the SVG was loaded.
SVGLoad	An SVG document has been loaded and parsed.
SVGResize	An SVG document is being resized.
SVGScroll	An SVG document is being scrolled.
SVGUnload	An SVG document has been removed from a window or frame.
SVGZoom	An SVG document is being zoomed.
timeout	
timeupdate	The time indicated by the currentTime attribute has been updated.

touchcancel	A touch point has been disrupted in an implementation-specific manners (too many touch points for example).
touchend	A touch point is removed from the touch surface.
touchenter	A touch point is moved onto the interactive area of an element.
touchleave	A touch point is moved off the interactive area of an element.
touchmove	A touch point is moved along the touch surface.
touchstart	A touch point is placed on the touch surface.
transitionend	A CSS transition has completed.
unload	The document or a dependent resource is being unloaded.
updateready	The resources listed in the manifest have been newly redownloaded, and the script can use swapCache() to switch to the new cache.
upgradeneeded	An attempt was made to open a database with a version number higher than its current version. A versionchange transaction has been created.
userproximity	Fresh data is available from a proximity sensor (indicates whether the nearby object is near the device or not).
versionchange	A versionchange transaction completed.
visibilitychange	The content of a tab has become visible or has been hidden.
volumechange	The volume has changed.
waiting	Playback has stopped because of a temporary lack of data.
wheel	A wheel button of a pointing device is rotated in any direction.

Anexo II: Crear eventos personalizados

Aquí se muestra cómo crear y activar eventos DOM. Estos eventos son comúnmente llamados eventos sintéticos, a diferencia de los eventos disparados por el navegador.

Los eventos pueden ser creados con el constructor de eventos de la siguiente manera:

```
var event = new Event('build'); // Escucha para el evento.

// Disparar event:

elem.addEventListener('build', function (e) { ... }, false);

elem.dispatchEvent(event); // Provoca que se ejecute el evento
```

El código de ejemplo de arriba usa el método ***EventTarget.dispatchEvent()*** el cual crea un nuevo evento sobre **elem** llamado **buld** y que ejecuta la función anónima indicada**.**

Para añadir más datos al evento, existe la interfaz ***CustomEvent***. Con este método podremos añadir propiedades a **event**. El segundo parámetro del CustomEvent es un **object** donde podremos especificar varios detalles en relación al comportamiento o contenido del evento. Por ejemplo, la propiedad más importante es **detail**, la cual se puede utilizar para pasar los datos personalizados. Por ejemplo, el evento se puede crear de la siguiente manera:

```
var event = new CustomEvent('build', {'detail': elem.dataset.time});
```

Esto permitirá tener acceso a los datos adicionales en el escuchador de eventos (event listener):

```
function eventHandler(e) {
  console.log ('The time is: ' + e.detail);
}
// Otro ejemplo:
const MessageEvent = new CustomEvent("message", {
  detail: {
    from: "Pepe",
    message: "Hello!"
  },
});
```

Anexo II: Eventos touch.

Desde la aparición de HTML5 y CSS3, todo ha cambiado y JavaScript lo ha hecho, dando un amplio soporte a dispositivos móviles como smartphones y tablets, además de servir como pilar de nuevos y asombrosos elementos HTML como el CANVAS, donde se construye cualquier cantidad de gráficos a través de JavaScript, pero en estos momentos lo que nos interesa de las nuevas herramientas del popular lenguaje web del lado del cliente, es el soporte que da al manejo de eventos “Touch”, extendiendo las APIS que tienen como tarea manejar todo lo referente a este tema.

Los dispositivos táctiles trabajan sin ningún problema en cualquier aplicación web, sus navegadores móviles hacen que los eventos de ratón como el clic, terminen por convertirse en un evento Touch, por eso, al hacer una aplicación web, ésta funciona, aunque sea procesando eventos del ratón. Ahora se han agregado a JavaScript una interesante gama de eventos Touch que, a pesar de ser solo de tres tipos, podemos hacer una combinación de los mismos, y así se obtienen mejores resultados.

Estos tipos de eventos son:

- **touchstart**: Este se genera al hacer cualquier toque a la pantalla, sin importar su duración o movimientos realizados.
- **touchend**: Este se ejecuta una vez se deja de tocar la pantalla o el objeto que tiene asignado el manejador de eventos.
- **touchmove**: Este es ejecutado una vez se desliza o desplaza el dedo el usuario, por encima de la pantalla u objeto que está siendo controlado a través del manejador de eventos.

Además, tenemos los siguientes componentes:

- **TouchEvent:** Representa un evento que ocurre cuando el estado de los toques en la superficie cambia.
- **Touch:** Representa un único punto de contacto entre el usuario y la superficie táctil.
- **TouchList:** Representa varios puntos de toque: esto se utiliza cuando el usuario tiene, por ejemplo, varios dedos en la superficie al mismo tiempo.
- **DocumentTouch:** Contiene varios métodos para crear objetos de Touch y TouchList.

Herramientas adicionales de los eventos Touch

Hasta ahora ya sabemos que contamos con tres tipos de eventos Touch, éstos, a su vez, tienen una serie de elementos que complementan todo el manejo y procesamientos que se generan tras la interacción de un usuario a través de un dispositivo táctil.

Es importante mencionar que cada evento Touch posee una lista de propiedades en común que vendrían siendo el complemento del que se hablaba anteriormente. Hay tres propiedades que están ligadas de forma directa al Touch, que son:

- **touches**: Es una lista de todos toques que se han generado en la pantalla, tiene poca utilidad y suele ser poco usada.
- **targetTouches**: Éste guarda una lista de la cantidad del evento que se ha generado en un elemento del DOM.
- **changedTouches**: Éste guarda una lista de todos cambios que se producen hasta llegar al evento Touch, por ejemplo, en un touchsend puede haber un touchstart y un touchmove.

Hay otro grupo de propiedades encargado de guardar información sobre el evento, las cuales son:

- **identifier**: Un número único que identifica de forma única cada toque generado durante una sesión.
- **target**: El elemento del DOM en donde se generó el evento.
- **client/ page/ screen**: Información de la pantalla, relevante sobre acciones que genera el evento.
- **radius coordinates and rotationAngle**: Describe una aproximación de las elipses generadas.

Procedimiento para asignar un evento Touch

Los eventos Touch de JavaScript no cambian para nada el esquema que normalmente se usa para crear manejadores en este lenguaje, así que ahora podemos asignar a cualquier elemento del DOM uno o varios de los tres eventos mencionados anteriormente.

Para facilitar un poco la explicación, veamos algo de código sencillo, empezaremos por obtener la referencia a través de métodos del DOM.

```
//Obtenemos el elemento con el que vamos a trabajar

var elementoTouch= document.getElementById("areaTactil");

//posteriormente asignamos el manejador de eventos lo cual se hace de manera convencional.

elementoTouch.addEventListener('touchstart', function(event) {
```

```
//Comprobamos si hay varios eventos del mismo tipo
  if (event.targetTouches.length == 1) {
    var touch = event.targetTouches[0];
    // con esto solo se procesa UN evento touch
    alert(" se ha producido un touchstart en las siguientes cordenas: X " + touch.pageX + " en Y " + touch.pageY);
  }
}, false);
```

En este caso se está manejando un solo evento. Para eventos multi-touch visitar las siguientes webs:

- https://developer.mozilla.org/es/docs/DOM/Touch_events
- https://developer.mozilla.org/en-US/docs/Web/API/Touch_events/Using_Touch_Events
- http://w3schools-fa.ir/jsref/obj_touchevent.html
