MODELO DE EVENTOS II

1. El flujo de eventos

Además de los eventos básicos que se han visto, los navegadores incluyen un mecanismo relacionado llamado **flujo de eventos** o "**event flow**". El flujo de eventos permite que **varios elementos diferentes** puedan **responder** a un **mismo evento**.

Si en una página HTML se define un elemento `<div>` con un botón en su interior, cuando el usuario pulsa sobre el botón, el navegador permite asignar una función de respuesta al botón, otra función de respuesta al `<div>` que lo contiene y otra función de respuesta a la página completa. De esta forma, **un solo evento** (la pulsación de un botón) provoca la **respuesta** de **tres elementos** de la página (incluyendo la propia página).

El **orden en el que se ejecutan los eventos** asignados a cada elemento de la página es lo que constituye el **flujo** de **eventos**. Además, existen muchas **diferencias** en el flujo de eventos de **cada navegador**.

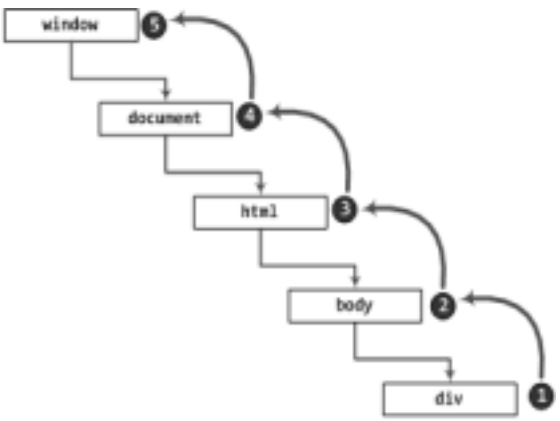
Event bubbling

En este modelo de flujo de eventos, el **orden** que se sigue es **desde** el **elemento más específico hasta** el **elemento menos específico**.

Utilizamos como **ejemplo** la siguiente página HTML:

```
<html onclick="procesaEvento()">
 <head><title>Ejemplo de flujo de eventos</title></head>
<body onclick="procesaEvento()">
 <div onclick="procesaEvento()">Haz clic aqui</div>
 </body>
</html>
```

Cuando se pulsa sobre el texto "Haz clic aquí" que se encuentra dentro del , se ejecutan los siguientes eventos en el orden que muestra el siguiente esquema:



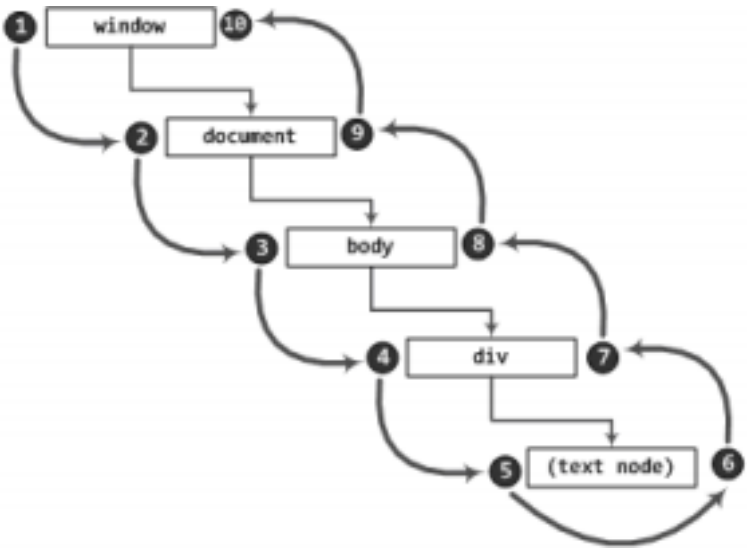
Event capturing

En ese otro modelo, el flujo de eventos se define desde el elemento menos específico hasta el elemento más específico. En otras palabras, el mecanismo definido es justamente el **contrario al "event bubbling"**.

Eventos DOM

El flujo de eventos definido en la especificación **DOM soporta** tanto el **bubbling** como el **capturing**, pero el "**event capturing**" **se ejecuta en primer lugar**. Los dos flujos de eventos recorren todos los objetos DOM desde el objeto document hasta el elemento más específico y viceversa. Además, la mayoría de navegadores que implementan los estándares, continúan el flujo hasta el objeto window.

El **flujo de eventos DOM** del ejemplo anterior se muestra a continuación:



El **elemento más específico** del flujo de eventos no es el `<div>` que desencadena la ejecución de los eventos, sino el **nodo** de tipo **TextNode** que contiene el `<div>`. El hecho de combinar los dos flujos de eventos, provoca que el nodo más específico pueda ejecutar dos eventos de forma consecutiva.

2. Handlers y listeners

Los "**event handlers**" o **manejadores de eventos**, son las **funciones que responden a los eventos** que se producen. Además, se vieron **tres formas de definir los manejadores de eventos** para el modelo básico de eventos:

- Código **JavaScript dentro** de un **atributo** del propio **elemento HTML** • Definición del evento en el propio elemento HTML pero el **manejador** es una **función externa**
- **Manejadores semánticos asignados mediante DOM** sin necesidad de modificar el código HTML de la página

Cualquiera de estos tres modelos funciona correctamente en todos los navegadores disponibles en la actualidad. Las **diferencias** entre **navegadores** surgen cuando se define **más de un manejador de eventos para un mismo evento** de un elemento. La forma de asignar y "desasignar" manejadores múltiples depende completamente del navegador utilizado.

Manejadores de eventos de Internet Explorer < IE9

La familia de navegadores de Internet Explorer en versiones inferiores a la 8 emplean los métodos **attachEvent()** y **detachEvent()** para asociar y desasociar manejadores de eventos. **Todos los elementos y el objeto window disponen de estos métodos**.

Los métodos requieren **dos parámetros**: el **nombre del evento** que se quiere manejar y una referencia a la **función encargada de procesar el evento**. El nombre del evento se debe indicar con el **prefijo on incluido**, como muestra el siguiente **ejemplo**:

```
function muestraMensaje() {
 alert("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.attachEvent("onclick", muestraMensaje);

// Más adelante se decide desasociar la función del
evento elDiv.detachEvent("onclick", muestraMensaje);
```

Asociar más de una función al mismo evento es igual de sencillo:

```
function muestraMensaje() {
 alert("Has pulsado el ratón");
}

function muestraOtroMensaje() {
 alert("Has pulsado el ratón y por eso se muestran estos mensajes");
}

var elDiv = document.getElementById("div_principal");
elDiv.attachEvent("onclick", muestraMensaje);
elDiv.attachEvent("onclick", muestraOtroMensaje);
```

Si el usuario hace clic sobre el , **se muestran los dos mensajes de aviso** que se han asignado al evento.

Se pueden **mezclar** incluso **diferentes técnicas** para asociar múltiples manejadores de eventos a un mismo evento de un elemento. El siguiente ejemplo utiliza la asignación semántica de manejadores para asignar el primer manejador y la función attachEvent() para asignar el segundo manejador. El resultado final es igual al del ejemplo anterior:

```
var elDiv = document.getElementById("div_principal");

elDiv.onclick = muestraMensaje;
```

```
elDiv.attachEvent("onclick", muestraOtroMensaje);
```

Sin embargo, **no es posible asignar múltiples eventos mediante las propiedades de DOM**:

```
var elDiv = document.getElementById("div_principal");
elDiv.onclick = muestraMensaje;
elDiv.onclick = muestraOtroMensaje;
```

Manejadores de eventos de DOM (2)

La especificación DOM define otros dos métodos similares a los disponibles para Internet Explorer y denominados **addEventListener()** y **removeEventListener()** para **asociar y desasociar manejadores de eventos**.

La principal diferencia entre estos métodos y los anteriores es que en este caso se **requieren tres parámetros**:

- El **nombre** del "**event listener**"
- Una referencia a la **función** encargada de procesar el evento
- El tipo de **flujo** de **eventos** al que se aplica. Es opcional.

El **primer argumento** no es el nombre completo del evento como sucede en el modelo de Internet Explorer, sino que **se debe eliminar el prefijo on**. En otras palabras, si en Internet Explorer se utilizaba el nombre **onclick**, ahora se debe utilizar **click**.

Si el **tercer parámetro** es **true**, el manejador se emplea en la fase de **capture**. Si el tercer parámetro es **false**, el manejador se asocia a la fase de **bubbling**.

A continuación, se muestran los **ejemplos** anteriores empleando los métodos definidos por **DOM**:

```
function muestraMensaje() {
 alert("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, false);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje,
false);
```

Asociando **múltiples funciones a un único evento**:

```
function muestraMensaje() {
 alert("Has pulsado el ratón");
}


function muestraOtroMensaje() {
```

```
 alert("Has pulsado el ratón y por eso se muestran estos mensajes");
}

var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, true);
elDiv.addEventListener("click", muestraOtroMensaje, true);
```

Si se asocia una función a un flujo de eventos determinado, esa función sólo se puede **desasociar en el mismo tipo de flujo de eventos**. Si se considera el siguiente ejemplo:

```
function muestraMensaje() {
 alert("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, false);

// Más adelante se decide desasociar la función al
evento elDiv.removeEventListener("click",
muestraMensaje, true);
```

La última instrucción intenta desasociar la función muestraMensaje en el flujo de eventos de capture, mientras que al asociarla, se indicó el **flujo de eventos de bubbling**. Aunque la ejecución de la aplicación no se detiene y no se produce ningún error, la última instrucción **no tiene ningún efecto**.

3. El objeto event

Cuando se produce un evento, no es suficiente con asignarle una función responsable de procesar ese evento. Normalmente, la función que procesa el evento necesita **información relativa al evento producido**: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.

El **objeto event** es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que **se crea automáticamente** cuando se **produce un evento** y que **se destruye de forma automática** cuando se han **ejecutado todas las funciones asignadas** al evento.

Para las versiones **Internet Explore (< IE9)** permite el acceso al objeto event a través del objeto **window**:

```
elDiv.onclick = function() {
 var elEvento = window.event;
}
```

El **estándar DOM** especifica que el **objeto event** es el **único parámetro que se debe pasar a las funciones encargadas de procesar los eventos**. Por tanto, en los navegadores que siguen los estándares, se puede acceder al objeto event **a través del array de los argumentos** de la función:

```
elDiv.onclick = function() {
 var elEvento = arguments[0];
}
```

También es posible **indicar el nombre argumento de forma explícita**:

```
elDiv.onclick = function(elEvento) {
 ...
}
```

El funcionamiento de los navegadores que siguen los estándares puede parecer "mágico", ya que en **la declaración de la función se indica que tiene un parámetro**, pero **en la aplicación no se pasa ningún parámetro** a esa función. En realidad, los navegadores que siguen los estándares **crean automáticamente ese parámetro y lo pasan siempre a la función encargada de manejar el evento**.

Propiedades y métodos

La siguiente tabla recoge las propiedades definidas para el objeto event en los navegadores que siguen los estándares

Propiedad/ Método	Devuelve	Descripción
altKey	Boolean	Devuelve true si se ha pulsado la tecla ALT y false en otro caso
button	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: 0 – Ningún botón pulsado 1 – Se ha pulsado el botón izquierdo 2 – Se ha pulsado el botón derecho 3 – Se pulsan a la vez el botón izquierdo y el derecho 4 – Se ha pulsado el botón central 5 – Se pulsan a la vez el botón izquierdo y el central 6 – Se pulsan a la vez el botón derecho y el central 7 – Se pulsan a la vez los 3 botones
cancelable	Boolean	Indica si el evento se puede cancelar
cancelBubble	Boolean	Si se establece un valor true, se detiene el flujo de eventos de tipo bubbling. OBSOLETO!
charCode	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
clientX	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
clientY	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
ctrlKey	Boolean	Devuelve true si se ha pulsado la tecla CTRL y false en otro caso

keyCode	Número entero	En el evento keypress, indica el carácter de la tecla pulsada. En los eventos keydown y keyup indica el código numérico de la tecla pulsada
offsetX	Número entero	Coordenada X de la posición del ratón respecto del elemento que origina el evento
offsetY	Número entero	Coordenada Y de la posición del ratón respecto del elemento que origina el evento
repeat	Boolean	Devuelve true si se está produciendo el evento keydown de forma continuada y false en otro caso
returnValue	Boolean	Se emplea para cancelar la acción predefinida del evento
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
shiftKey	Boolean	Devuelve true si se ha pulsado la tecla SHIFT y false en otro caso
target	Element	El elemento que origina el evento
type	Cadena de texto	El nombre del evento
…		
Consulta: https://www.w3schools.com/jsref/dom_obj_event.asp		

La **tecla META** es una tecla especial que se encuentra en algunos teclados de ordenadores muy antiguos. Actualmente, en los ordenadores **tipo PC** se asimila a la tecla **Alt** o a la tecla de **Windows**, mientras que en los ordenadores **tipo Mac** se asimila a la tecla **Command**.

Ejemplos de uso:

```
function procesaEvento(elEvento) {
 if(elEvento.type == "click") {
 alert("Has pulsado el ratón");
 }
 else if(elEvento.type == "mouseover") {
 alert("Has movido el ratón");
 }
}

elDiv.onclick = procesaEvento;
```

```
elDiv.onmouseover = procesaEvento;
```

Mientras que el manejador del evento incluye el prefijo on en su nombre, el **tipo de evento devuelto** por la propiedad type **prescinde de** ese **prefijo**. Por eso en el ejemplo anterior se compara su valor con **click** y **mouseover** y no con **onclick** y **onmouseover**.

Lla propiedad **keyCode** devuelve el **código correspondiente al carácter de la tecla que se ha pulsado**. La tecla pulsada **no siempre representa un carácter alfanumérico**. Cuando se pulsa la tecla ENTER por ejemplo, se obtiene el código 13. La barra espaciadora se corresponde con el código 32 y la tecla de borrado tiene un código igual a 8.

Una forma más inmediata de comprobar si se han pulsado algunas **teclas especiales**, es utilizar las propiedades **shiftKey**, **altKey** y **ctrlKey**.

Para obtener la **posición del ratón respecto de la parte visible de la ventana**, se emplean las propiedades **clientX** y **clientY**. De la misma forma, para obtener la **posición del puntero del ratón respecto de la pantalla completa**, se emplean las propiedades **screenX** y **screenY**.

Obtener el elemento que origina el evento. Si un elemento tiene asignado un evento onclick, al pulsar con el ratón el interior del se origina un evento cuyo objetivo es el elemento .

```
// Navegadores que siguen los estandares
var objetivo = elEvento.target;
```

Obtención del **carácter correspondiente a la tecla pulsada**. Cada tecla pulsada tiene asociados dos códigos diferentes:

- el código de la **tecla que ha sido presionada**: código de tecla interno para JavaScript
- el código que se refiere al **carácter pulsado**: coincide con el código ASCII del carácter

De esta forma, la **letra a** tiene un **código interno igual a 65** y un **código ASCII de 97**. Por otro lado, la **letra A** tiene un **código interno también de 65** y un **código ASCII de 95**.

```
function manejador(elEvento) {
 var evento = elEvento || window.event;
 alert("["+evento.type+"] El código de la tecla pulsada es " +
evento.keyCode);
}
document.onkeyup = manejador;
document.onkeydown = manejador;
```

En este caso, si se carga la página en cualquier navegador y se pulsa por ejemplo la tecla a, se muestra el siguiente mensaje:

La gran **diferencia** se produce al intentar **obtener el carácter que se ha pulsado**, en este caso la letra a. Para obtener la letra, en primer lugar se debe obtener su código ASCII. Como se ha comentado, **en Internet Explorer** el valor de la propiedad **keyCode** en el evento **onkeypress** es igual al carácter ASCII:

```
function manejador() {
 var evento = window.event;

 // Internet Explorer
 var codigo = evento.keyCode;
}

document.onkeypress = manejador;
```

Una vez obtenido el código en cada navegador, se debe utilizar la función **String.fromCharCode() para obtener el carácter cuyo código ASCII se pasa como parámetro**. Por tanto, la solución completa para obtener la tecla pulsada **en cualquier navegador** es la siguiente:

```
function manejador(elEvento) {
 var evento = elEvento || window.event;
 var codigo = evento.charCode || evento.keyCode;
 var caracter = String.fromCharCode(codigo);
}

document.onkeypress = manejador;
```

Una de las propiedades más interesantes es la posibilidad de **impedir que se complete el comportamiento normal de un evento**. En otras palabras, con JavaScript es posible **no mostrar ningún carácter cuando se pulsa una tecla**, **no enviar un formulario después de pulsar el botón de envío**, **no cargar ninguna página al pulsar un enlace**, etc. El método avanzado de impedir que un evento ejecute su acción asociada:

```
// Navegadores que siguen los estandares
elEvento.preventDefault();
```

En el **modelo básico de eventos** también es posible **impedir el comportamiento por defecto** de algunos eventos. Si por ejemplo en un elemento **<textarea>** se indica el siguiente manejador de eventos:

```
<textarea onkeypress="return false;"></textarea>
```

En el **<textarea>** anterior **no será posible escribir ningún carácter**, ya que el manejador de eventos devuelve false y ese es el valor necesario para impedir que se termine de ejecutar el evento y por tanto para evitar que la letra se escriba.

Así, es posible definir **manejadores** de eventos que **devuelvan true o false en función de algunos parámetros**. Por ejemplo se puede diseñar un **limitador del número de caracteres** que se pueden escribir en un <textarea>:

```
function limita(maximoCaracteres) {
```

```
var elemento = document.getElementById("texto");
 if(elemento.value.length >= maximoCaracteres ) {
 return false;
 }
 else {
 return true;
 }
}

<textarea id="texto" onkeypress="return
```

limita(100);"></textarea> El funcionamiento del **ejemplo** anterior:

1. Se utiliza el evento **onkeypress** para **controlar si la tecla se escribe o no**. 2. En el manejador del evento **se devuelve el valor devuelto por** la función externa **limita()** a la que se **pasa como parámetro el valor 100**.
3. Si el valor devuelto por **limita()** es **true**, el **evento se produce de forma normal** y el **carácter se escribe** en el **<textarea>**. Si el valor devuelto por **limita()** es **false**, el **evento no se produce** y por tanto **el carácter no se escribe** en el **<textarea>**.
4. La función **limita()** devuelve true o false después de **comprobar si el número de caracteres** del **<textarea>** es **superior o inferior al máximo** número de caracteres que se le ha **pasado como parámetro**.

El **objeto event** también permite **detener completamente la ejecución del flujo normal de eventos**:

```
// Navegadores que siguen los estandares
elEvento.stopPropagation();
```

Al detener el flujo de eventos pendientes, **se invalidan y no se ejecutan los eventos que restan desde ese momento hasta que se recorren todos los elementos pendientes hasta el elemento window**.

4. Tipos de eventos

La **lista completa de eventos** que se pueden generar en un navegador se puede dividir en **cuatro grandes grupos**. La especificación de DOM define los siguientes grupos:

- **Eventos de ratón**: se originan cuando el usuario emplea el **ratón** para realizar algunas acciones.
- **Eventos de teclado**: se originan cuando el usuario pulsa sobre cualquier **tecla** de su teclado.
- **Eventos HTML**: se originan cuando se producen cambios en la ventana del **navegador** o cuando se producen ciertas **interacciones** entre el **cliente** y el **servidor**.
- **Eventos DOM**: se originan cuando se produce un cambio en la **estructura DOM** de la página. También se denominan "eventos de mutación".

Eventos de ratón

Los eventos de ratón son, con mucha diferencia, los más empleados en las aplicaciones web. Los eventos que se incluyen en esta clasificación son los siguientes:

Evento	Descripción
click	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla ENTER
dblclick	Se produce cuando se pulsa dos veces el botón izquierdo del ratón
mousedown	Se produce cuando se pulsa cualquier botón del ratón
mouseout	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario mueve el puntero a un lugar fuera de ese elemento
mouseover	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento
mouseup	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado
mousemove	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento
contextmenu	Se produce cuando se pulsa el boton derecho

Todos los elementos de las páginas soportan los eventos de la tabla anterior.

Propiedades

El **objeto event** contiene las siguientes propiedades para los eventos de ratón:

- Las **coordenadas del ratón** (todas las coordenadas diferentes relativas a los distintos elementos)
- La **propiedad type**
- La **propiedad target**
- Las **propiedades shiftKey**, **ctrlKey**, **altKey** y **metaKey**
- La **propiedad button** (sólo en los eventos mousedown, mousemove, mouseout, mouseover y mouseup)

En el evento **mouseout**, **relatedTarget** apunta al **elemento al que se ha movido el ratón**. En el evento **mouseover**, **relatedTarget** apunta al **elemento desde el que se ha movido el puntero del ratón**.

Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: mousedown, mouseup, click. Por tanto, la secuencia de eventos necesaria para llegar al doble click llega a ser tan compleja como la siguiente: mousedown, mouseup, click, mousedown, mouseup, click, dblclick.

Eventos de teclado

Los eventos que se incluyen en esta clasificación son los siguientes:

Evento	Descripción
keydown	Se produce cuando se pulsa cualquier tecla del teclado. También se produce de forma continua si se mantiene pulsada la tecla
keypress	Se produce cuando se pulsa una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta telas como SHIFT, ALT, etc.). También se produce de forma continua si se mantiene pulsada la tecla
keyup	Se produce cuando se suelta cualquier tecla pulsada

Propiedades

El objeto event contiene las siguientes propiedades para los eventos de teclado:

- La **propiedad keyCode**
- La **propiedad charCode** (sólo DOM)
- La **propiedad target**
- Las **propiedades shiftKey**, **ctrlKey**, **altKey** y **metaKey**

Cuando se pulsa una tecla correspondiente a un **carácter alfanumérico**, se produce la siguiente **secuencia de eventos**:

1. **keydown**
2. **keypress**
3. **keyup**

Cuando se pulsa **otro tipo de tecla**, se produce la siguiente secuencia de eventos:

1. **keydown**
2. **keyup**

Si se **mantiene pulsada la tecla**, en el **primer caso** se **repiten de forma continua los eventos keydown y keypress** y en el **segundo caso**, se **repite el evento keydown de forma continua**.

Eventos HTML

Los eventos HTML definidos se recogen en la siguiente tabla:

Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo. En el elemento <img> cuando se carga por completo la imagen. En el elemento <object> cuando se carga el objeto
abort	Se produce en un elemento <object> cuando el usuario detiene la descarga del elemento antes de que haya terminado
error	Se produce en el objeto window cuando se produce un error de JavaScript. En el elemento <img> cuando la imagen no se ha podido cargar por completo y en el elemento <object> cuando el elemento no se carga correctamente
select	Se produce cuando se seleccionan varios caracteres de un cuadro de texto (<input> y <textarea>)
change	Se produce cuando un cuadro de texto (<input> y <textarea>) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento <select>
submit	Se produce cuando se pulsa sobre un botón de tipo submit (<input type="submit">)
reset	Se produce cuando se pulsa sobre un botón de tipo reset (<input type="reset">)
resize	Se produce en el objeto window cuando se redimensiona la ventana del navegador
scroll	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento <body> contiene la barra de scroll de la página completa
focus	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco
blur	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco

https://www.w3schools.com/tags/ref_eventattributes.asp

Uno de los eventos más utilizados es el **evento load**, ya que todas las **manipulaciones que se realizan mediante DOM requieren que la página esté cargada por completo** y por tanto, el árbol DOM se haya construido completamente.

El elemento **<body>** define las propiedades **scrollLeft** y **scrollTop** que se pueden emplear junto con el evento **scroll**.

Ver ejemplo:https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_element_scrollleft

Eventos DOM

Aunque los eventos de este tipo son parte de la especificación oficial de DOM, aún no han sido implementados en todos los navegadores. La siguiente tabla recoge los eventos más importantes de este tipo:

Evento	Descripción
DOMSubtreeModified	Se produce cuando se añaden o eliminan nodos en el subárbol de un documento o elemento
DOMNodeInserted	Se produce cuando se añade un nodo como hijo de otro nodo
DOMNodeRemoved	Se produce cuando se elimina un nodo que es hijo de otro nodo
DOMNodeRemovedFromDocument	Se produce cuando se elimina un nodo del documento
DOMNodeInsertedIntoDocument	Se produce cuando se añade un nodo al documento

Ver ejemplos: http://help.dottoro.com/ljmcxjla.php