

<codoa  
codoo/>

1 Manipulacion de DOM

2 Eventos

Agencia de  
Aprendizaje  
a lo largo  
de la vida

## ¿Qué es el DOM?

El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

Una página web es un documento. Éste documento puede exhibirse en la ventana de un navegador o también como código fuente HTML. Pero, en los dos casos, es el mismo documento. El modelo de objeto de documento (DOM) proporciona otras formas de presentar, guardar y manipular este mismo documento. El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript.

El W3C DOM estándar forma la base del funcionamiento del DOM en muchos navegadores modernos. Varios navegadores ofrecen extensiones más allá del estándar W3C, hay que ir con extremo cuidado al utilizarlas en la web, ya que los documentos pueden ser consultados por navegadores que tienen DOMs diferentes.

Por ejemplo, el DOM de W3C especifica que el método `getElementsByTagName` en el código de abajo debe devolver una lista de todos los elementos `<p>` del documento:

```
paragraphs = document.getElementsByTagName ("p");
```

```
// paragraphs[0] es el primer elemento <p>
```

```
// paragraphs[1] es el segundo elemento <p>, etc.
```

```
alert (paragraphs [0].nodeName);
```





Todas las propiedades, métodos y eventos disponibles para la manipulación y la creación de páginas web está organizado dentro de objetos.

Un ejemplo: el objeto document representa al documento mismo, el objeto table hace funcionar la interfaz especial HTMLTableElement del DOM para acceder a tablas HTML, y así sucesivamente.

## DOM y JavaScript

El DOM no es un lenguaje de programación pero sin él, el lenguaje JavaScript no tiene ningún modelo o noción de las páginas web, de la páginas XML ni de los elementos con los cuales es usualmente relacionado. Cada elemento -"el documento íntegro, el título, las tablas dentro del documento, los títulos de las tablas, el texto dentro de las celdas de las tablas"- es parte del modelo de objeto del documento para cada documento, así se puede acceder y manipularlos utilizando el DOM y un lenguaje de escritura, como JavaScript.

En el comienzo, JavaScript y el DOM estaban herméticamente enlazados, pero después se desarrollaron como entidades separadas. El contenido de la página es almacenado en DOM y el acceso y la manipulación se hace vía JavaScript, podría representarse aproximadamente así:

API(web o página XML) = DOM + JS(lenguaje de script)

El DOM fue diseñado para ser independiente de cualquier lenguaje de programación particular, hace que la presentación estructural del documento sea disponible desde un simple y consistente API.

## ¿Cómo se accede al DOM?

No se tiene que hacer nada especial para empezar a utilizar el DOM. Los diferentes navegadores tienen directrices DOM distintas, y éstas directrices tienen diversos grados de conformidad al actual estándar DOM, pero todos los navegadores web usan el modelo de objeto de documento para hacer accesibles las páginas web al script.

Cuando se crea un script —esté en un elemento `<SCRIPT>` o incluido en una página web por la instrucción de cargar un script— inmediatamente está disponible para usarlo con el API, accediendo así a los elementos `document` o `window`, para manipular el documento mismo o sus diferentes partes, las cuales son los varios elementos de una página web. La programación DOM hace algo tan simple como lo siguiente, lo cual abre un mensaje de alerta usando la función `alert()` desde el objeto `window`, o permite métodos DOM más sofisticados para crear realmente un nuevo contenido, como en el largo ejemplo de más abajo.

```
<body onload="window.alert('Bienvenido a mi página!');">
```

Aparte del elemento `<script>` en el cual JavaScript es definido, el ejemplo siguiente muestra la función a ejecutar cuando el documento se está cargando (y que el DOM completo es disponible para su uso). Esta función crea un nuevo elemento `H1`, le pone texto y después lo agrega al árbol del documento:

```
<html>  
  <head>  
    <script>
```



```
// ejecuta esta función cuando se cargue el documento  
window.onload = function() {
```

```
    // crea dinámicamente un par de elementos HTML en una página vacía  
    var heading = document.createElement("h1");  
    var heading_text = document.createTextNode("el texto que desee");  
    heading.appendChild(heading_text);  
    document.body.appendChild(heading);  
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

## Tipos de datos importantes

Esta parte intenta describir, de la manera más simple posible, los diferentes objetos y tipos. Pero hay que conocer una cantidad de tipos de datos diferentes que son utilizados por el API. Para simplificarlo, los ejemplos de sintaxis en esta API se refieren a nodos como `elements`, a una lista de nodos como `nodeLists` (o simples elementos) y a nodos de atributo como `attributes`.

La siguiente tabla describe brevemente estos tipos de datos.

### document

Cuando un miembro devuelve un objeto del tipo `document` (por ejemplo, la propiedad `ownerDocument` de un elemento devuelve el documento "document" al cual pertenece), este objeto es la raíz del objeto documento en sí mismo. El capítulo La referencia al documento (document) de DOM lo explica con más detalles.

## element

`element` se refiere a un elemento o a un nodo de tipo de elemento "element" devuelto por un miembro del API de DOM. Dicho de otra manera, por ejemplo, el método `document.createElement()` devuelve un objeto referido a un nodo, lo que significa que este método devuelve el elemento que acaba de ser creado en el DOM. Los objetos `element` ponen en funcionamiento a la interfaz `Element` del DOM y también a la interfaz de nodo "Node" más básica, las cuales son incluidas en esta referencia.

## nodeList

Una "nodeList" es una serie de elementos, parecido a lo que devuelve el método `document.getElementsByTagName()`. Se accede a los items de la `nodeList` de cualquiera de las siguientes dos formas:

`list.item (1)`

`lista [1]`

Ambas maneras son equivalentes. En la primera, `item()` es el método del objeto `nodeList`. En la última se utiliza la típica sintaxis de acceso a listas para llegar al segundo ítem de la lista.

## attribute

Cuando un atributo ("attribute") es devuelto por un miembro (por ej., por el método `createAttribute()`), es una referencia a un objeto que expone una interfaz particular (aunque limitada) a los atributos. Los atributos son nodos en el DOM igual que los elementos, pero no suelen usarse así.



## NamedNodeMap

Un namedNodeMap es una serie, pero los ítems son accesibles tanto por el nombre o por un índice, este último caso es meramente una conveniencia para enumerar ya que no están en ningún orden en particular en la lista. Un NamedNodeMap es un método de ítem() por esa razón, y permite poner o quitar ítems en un NamedNodeMap

## Interfaces del DOM

Desde el punto de vista del programador web, es bastante indiferente saber que la representación del objeto del elemento HTML form toma la propiedad name desde la interfaz HTMLFormElement pero que las propiedades className se toman desde la propia interfaz HTMLElement. En ambos casos, la propiedad está sólo en el objeto form.

Pero puede resultar confuso el funcionamiento de la fuerte relación entre objetos e interfaces en el DOM, por eso intentaré hablar un poquito sobre las interfaces actuales en la especificación del DOM y de como se dispone de ellas.

## Interfaces y objetos

En algunos casos un objeto pone en ejecución a una sola interfaz. Pero a menudo un objeto toma prestada una tabla HTML (table) desde muchas interfaces diversas. El objeto table, por ejemplo, pone en funcionamiento una Interfaz especial del elemento table HTML, la cual incluye métodos como createCaption y insertRow. Pero como también es un elemento HTML, table pone en marcha

a la interfaz del Element descrita en el capítulo La referencia al elemento del DOM. Y finalmente, puesto que un elemento HTML es también, por lo que concierne al DOM, un nodo en el árbol de nodos que hace el modelo de objeto para una página web o XML, el elemento de table hace funcionar la interfaz más básica de Node, desde el cual deriva Element.

La referencia a un objeto table, como en el ejemplo siguiente, utiliza estas interfaces intercambiables sobre el objeto.

```
var table = document.getElementById("table");
var tableAttrs = table.attributes; // Node/interfaz Element for
(var i = 0; i < tableAttrs.length; i++) {

    // interfaz HTMLTableElement: atributo border
    if(tableAttrs[i].nodeName.toLowerCase() == "border")
        table.border = "1";
}

// interfaz HTMLTableElement: atributo summary
table.summary = "nota: borde aumentado";
```

## Interfaces esenciales en el DOM document y window

son objetos cuya interfaces son generalmente muy usadas en la programación de DOM. En término simple, el objeto window representa algo como podría ser el navegador, y el objeto document es la raíz del documento en sí. Element hereda de la interfaz genérica Node, y juntos, estas dos interfaces proporcionan muchos métodos y propiedades utilizables sobre los elementos individuales. Éstos elementos pueden igualmente tener interfaces específicas según el tipo de datos representados, como en el ejemplo anterior del objeto table. Lo siguiente es una breve lista de los APIS comunes en la web y en las páginas escritas en XML utilizando el DOM.

```
document.getElementById(id)
element.getElementsByTagName(name)
```



```
document.createElement(name)  
parentNode.appendChild(node) element.innerHTML
```

```
element.style.left  
element.setAttribute  
element.element.getAttribute  
element.addEventListener  
window.content window.onload  
window.dump window.scrollTo
```

## Probando el API del DOM

Ésta parte procura ejemplos para todas las interfaces usadas en el desarrollo web. En algunos casos, los ejemplos son páginas HTML enteras, con el acceso del DOM a un elemento de `<script>`, la interfaz necesaria (por ejemplo, botones) para la ejecución del script en un formulario, y también que los elementos HTML sobre los cuales opera el DOM se listen. Según el caso, los ejemplos se pueden copiar y pegar en un documento web para probarlos.

No es el caso donde los ejemplos son muchos más concisos. Para la ejecución de estos ejemplos que sólo demuestran la relación básica entre la interfaz y los elementos HTML, resulta útil tener una página de prueba en la cual las interfaces serán fácilmente accesibles por los scripts. La página siguiente proporciona en las cabeceras un elemento de script en el cual se pondrán las funciones para testar la interfaz elegida, algunos elementos HTML con atributos que se puedan leer, editar y también manipular, así como la interfaz web necesaria para llamar esas funciones desde el navegador.

Para probar y ver como trabajan en la plataforma del navegador las interfaces del DOM, esta página de prueba o una nueva similar son factibles. El contenido de la función test() se puede actualizar según la necesidad, para crear más botones o poner más elementos.

```
<html>
  <head>
    <title>Pruebas DOM</title>
    <script type="application/javascript">
      function setBodyAttr(attr, value){
        if (document.body) eval('document.body.'+attr+'="'+value+'"; else notSupported());
      }
    </script>
  </head>
  <body>
    <div style="margin: .5in; height: 400px;">
      <p><b><tt>texto</tt></b></p>
      <form>
        <select onChange="setBodyAttr('text',
          this.options[this.selectedIndex].value);">
          <option value="black">negro
          <option value="darkblue">azul oscuro
```



```
</select>
<p><b><tt>bgColor</tt></b></p>
<select onChange="setBodyAttr('bgColor',
this.options[this.selectedIndex].value);">
  <option value="white">blanco
  <option value="lightgrey">gris
</select>
<p><b><tt>link</tt></b></p>
<select onChange="setBodyAttr('link',
this.options[this.selectedIndex].value);">
<option value="blue">azul
  <option value="green">verde
</select> <small>
<a href="http://www.brownhen.com/dom_api_top.html" id="sample">
(sample link)</a></small><br>
</form>
<form>
  <input type="button" value="version" onclick="ver()" />
</form>
</div>
</body>
</html>
```

## Eventos

Los eventos son acciones u ocurrencias que suceden en el sistema que está programando y que el sistema le informa para que pueda responder de alguna manera si lo desea. Por ejemplo, si el usuario hace clic en un botón en una página web, es posible que desee responder a esa acción mostrando un cuadro de información. En este artículo, discutiremos algunos conceptos importantes que rodean los eventos y veremos cómo funcionan en los navegadores. Este no será un estudio exhaustivo; solo lo que necesitas saber en esta etapa.

En el caso de la Web, los eventos se desencadenan dentro de la ventana del navegador y tienden a estar unidos a un elemento específico que reside en ella — podría ser un solo elemento, un conjunto de elementos, el documento HTML cargado en la pestaña actual o toda la ventana del navegador.

Hay muchos tipos diferentes de eventos que pueden ocurrir, por ejemplo:

El usuario hace clic con el mouse sobre un elemento determinado o coloca el cursor sobre un elemento determinado.

El usuario presiona una tecla en el teclado.

El usuario cambia el tamaño o cierra la ventana del navegador. Una página web termina de cargar.

Un formulario se envía

Un video se reproduce, pausa o finaliza la reproducción. Un error ocurre.

Se deducirá de esto que hay muchos eventos a los que se puede responder. Puede consultarse la referencia completa de eventos en: <https://developer.mozilla.org/es/docs/Web/Events>

Un ejemplo simple

Veamos un ejemplo simple para explicar lo que queremos decir aquí. Ya has visto eventos y controladores de eventos en muchos de los ejemplos de este curso, pero vamos a recapitular solo para



consolidar nuestro conocimiento. En el siguiente ejemplo, tenemos un solo `<button>`, que cuando se presiona, hará que el fondo cambie a un color aleatorio:

`<button>Cambiar color</button>` El JavaScript se ve así:

```
const btn = document.querySelector('button'); function random(number) {  
return Math.floor(Math.random() * (number+1));  
}  
  
btn.onclick = function() {  
    const rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
    document.body.style.backgroundColor = rndCol;  
}
```

En este código, almacenamos una referencia al botón dentro de una variable llamada `btn`, usando la función `Document.querySelector()`. También definimos una función que devuelve un número aleatorio. La tercera parte del código es el controlador de eventos. La variable `btn` apunta a un elemento `<button>`, y este tipo de objeto tiene una serie de eventos que pueden activarse y, por lo tanto, los controladores de eventos están disponibles. Estamos escuchando el disparo del evento "click", estableciendo la propiedad del controlador de eventos `onclick` para que sea igual a una función anónima que contiene código que generó un color RGB aleatorio y establece el `<body>` color de fondo igual a este.

Este código ahora se ejecutará cada vez que se active el evento "click" en el elemento <button>, es decir, cada vez que un usuario haga clic en él.

## Diferentes formas de uso de eventos

Hay muchas maneras distintas en las que puedes agregar event listeners a los sitios web, que se ejecutara cuando el evento asociado se dispare. En esta sección, revisaremos los diferentes mecanismos y discutiremos cuales deberias usar..

## Propiedades de manejadores de eventos

Estas son las propiedades que existen, que contienen código de manejadores de eventos(Event Handler) que vemos frecuentemente durante el curso.. Volviendo al ejemplo de arriba:

```
var btn = document.querySelector("button"); btn.onclick = function() {  
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
  document.body.style.backgroundColor = rndCol;  
}
```

La propiedad onclick es la propiedad del manejador de eventos que está siendo usada en esta situación. Es esencialmente una propiedad como cualquier otra disponible en el botón (por ejemplo: btn.textContent, or btn.style), pero es de un tipo especial — cuando lo configura para ser igual a algún código, ese código se ejecutará cuando el evento se dispare en el botón.

También puede establecer la propiedad del controlador para que sea igual a un nombre de función con nombre (como vimos en Construya su propia función ). Lo siguiente funcionaría igual:

```
var btn = document.querySelector("button"); function bgChange() {  
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
  document.body.style.backgroundColor = rndCol;  
}
```

```
btn.onclick = bgChange;
```



Hay muchas propiedades de controlador de eventos diferentes disponibles. Hagamos un experimento.

En primer lugar, haga una copia local de `random-color-eventhandlerproperty.html` y ábralo en su navegador. Es solo una copia del ejemplo simple de color aleatorio con el que hemos estado jugando en este artículo. Ahora intente cambiar `btn.onclick` a los siguientes valores diferentes a su vez y observe los resultados en el ejemplo:

`btn.onfocus` y `btn.onblur`- El color cambiará cuando el botón esté enfocado y desenfocado (intente presionar pestaña a pestaña en el botón y apáguelo nuevamente). Estos se utilizan a menudo para mostrar información sobre cómo completar los campos del formulario cuando están enfocados, o mostrar un mensaje de error si un campo del formulario se acaba de completar con un valor incorrecto.

`btn.ondblclick` - El color cambiará solo cuando se haga doble clic en él.

`window.onkeypress`, `window.onkeydown`, `window.onkeyup`- El color cambiará cuando se pulsa una tecla del teclado. `keypress` se refiere a una pulsación general (botón hacia abajo y luego hacia arriba), mientras que `keydown` y `keyup` prefieren solo a las partes de la pulsación de tecla hacia

abajo y hacia arriba, respectivamente. Tenga en cuenta que no funciona si intenta registrar este controlador de eventos en el propio botón; hemos tenido que registrarlo en el objeto de ventana , que representa la ventana completa del navegador.

btn.onmouseover btn.onmouseout- El color cambiará cuando el puntero del mouse se mueva para que comience a desplazarse sobre el botón, o cuando deje de desplazarse sobre el botón y se mueva fuera de él, respectivamente.

Algunos eventos son muy generales y están disponibles en casi cualquier lugar (por ejemplo, un onclickcontrolador se puede registrar en casi cualquier elemento), mientras que algunos son más específicos y solo útiles en ciertas situaciones (por ejemplo, tiene sentido usar onplay solo en elementos específicos, como <video>).

Controladores de eventos en línea: no los use También puede ver un patrón como este en su código:

```
<button onclick="bgChange()">Press me</button>
function bgChange() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}
```

Nota : Puede encontrar el código fuente completo para este ejemplo en GitHub (también verlo ejecutándose en vivo ).

El primer método para registrar controladores de eventos que se encuentra en la Web involucró atributos HTML del controlador de eventos (también conocidos como controladores de eventos en línea ) como el que se muestra arriba: el valor del atributo es literalmente el código JavaScript que desea ejecutar cuando ocurre el evento. El ejemplo anterior invoca una función definida dentro de un <script>elemento en la misma página, pero también puede insertar JavaScript directamente dentro del atributo, por ejemplo:



```
<button onclick="alert('Hello, this is my old-fashioned event handler!');">Press me</button>
```

Encontrará equivalentes de atributos HTML para muchas de las propiedades del controlador de eventos; sin embargo, no debe utilizarlos, ya que se consideran una mala práctica. Puede parecer fácil usar un atributo de controlador de eventos si solo está haciendo algo realmente rápido, pero muy rápidamente se vuelven inmanejables e ineficientes.

Para empezar, no es una buena idea mezclar su HTML y su JavaScript, ya que se vuelve difícil de analizar - es mejor mantener su JavaScript en un solo lugar; si está en un archivo separado, puede aplicarlo a varios documentos HTML.

Incluso en un solo archivo, los controladores de eventos en línea no son una buena idea. Un botón está bien, pero ¿y si tuviera 100 botones? Tendría que agregar 100 atributos al archivo; muy rápidamente se convertiría en una pesadilla de mantenimiento. Con JavaScript, puede agregar fácilmente una función de controlador de eventos a todos los botones de la página, sin importar cuántos haya, usando algo como esto:

```
var buttons = document.querySelectorAll('button');
```

```
for (var i = 0; i < buttons.length; i++) {  
  buttons[i].onclick = bgChange;  
}
```

Tenga en cuenta que otra opción aquí sería utilizar el `forEach()` método integrado disponible en todos los objetos Array:

```
buttons.forEach(function(button) {  
  button.onclick = bgChange;  
});
```

Nota : Separar su lógica de programación de su contenido también hace que su sitio sea más amigable para los motores de búsqueda.

addEventListener () y removeEventListener ()

El último tipo de mecanismo de eventos se define en el Document Object Model (DOM) Nivel 2 Eventos , la cual provee los navegadores con una nueva función - addEventListener(). Esto funciona de manera similar a las propiedades del controlador de eventos, pero la sintaxis es obviamente diferente. Podríamos reescribir nuestro ejemplo de color aleatorio para que se vea así:

```
var btn = document.querySelector('button'); function bgChange() {  
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
  document.body.style.backgroundColor = rndCol;  
}
```

```
btn.addEventListener('click', bgChange);
```

Nota : Puede encontrar el código fuente completo para este ejemplo en GitHub (también verlo ejecutándose en vivo ).

Dentro de la addEventListener()función, especificamos dos parámetros: el nombre del evento para el que queremos registrar este controlador y el código que comprende la función del controlador que queremos ejecutar en respuesta a él. Tenga en cuenta que es perfectamente apropiado poner todo el código dentro de la addEventListener()función, en una función anónima, como esta:

```
btn.addEventListener('click', function() {  
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
  document.body.style.backgroundColor = rndCol;  
});
```



```
});
```

Este mecanismo tiene algunas ventajas sobre los mecanismos más antiguos discutidos anteriormente. Para empezar, hay una función de contraparte `removeEventListener()`, que elimina un oyente agregado previamente. Por ejemplo, esto eliminaría el conjunto de escuchas en el primer bloque de código de esta sección:

```
btn.removeEventListener('click', bgChange);
```

Esto no es significativo para programas pequeños y simples, pero para programas más grandes y complejos puede mejorar la eficiencia para limpiar los controladores de eventos antiguos que no se utilizan. Además, por ejemplo, esto le permite tener el mismo botón realizando diferentes acciones en diferentes circunstancias; todo lo que tiene que hacer es agregar / eliminar controladores de eventos según corresponda.

En segundo lugar, también puede registrar varios controladores para el mismo oyente. No se aplicarían los dos controladores siguientes:

```
myElement.onclick = functionA;  
myElement.onclick = functionB;
```

Como la segunda línea sobrescribiría el valor de `onclick` por la primera. Sin embargo, esto funcionaría:

```
myElement.addEventListener('click', functionA);  
myElement.addEventListener('click', functionB);
```

Ambas funciones ahora se ejecutarían cuando se haga clic en el elemento.

Además, hay una serie de otras funciones y opciones potentes disponibles con este mecanismo de eventos. Estos están un poco fuera del alcance de este artículo, pero si desea leer sobre ellos, eche un vistazo a las páginas de referencia `addEventListener()` y `removeEventListener()`.

¿Qué mecanismo debo utilizar?

De los tres mecanismos, definitivamente no debe usar los atributos del controlador de eventos HTML; estos están desactualizados y son una mala práctica, como se mencionó anteriormente.

Los otros dos son relativamente intercambiables, al menos para usos simples:

Las propiedades del controlador de eventos tienen menos potencia y opciones, pero una mejor compatibilidad entre navegadores (siendo compatibles desde Internet Explorer 8). Probablemente debería comenzar con estos a medida que aprende.

Los eventos DOM de nivel 2 (`addEventListener()`, etc.) son más potentes, pero también pueden volverse más complejos y tienen menos soporte (admitidos desde Internet Explorer 9). También debe experimentar con ellos y tratar de utilizarlos siempre que sea posible.

Las principales ventajas del tercer mecanismo son que puede eliminar el código del controlador de eventos si es necesario, utilizando `removeEventListener()`, y puede agregar varios oyentes del mismo tipo a los elementos si es necesario. Por ejemplo, puede llamar `addEventListener('click', function() { ... })` a un elemento varias veces, con diferentes funciones especificadas en el segundo argumento. Esto es imposible con las propiedades del controlador de eventos porque cualquier intento posterior de establecer una propiedad sobrescribirá los anteriores, por ejemplo:



```
element.onclick = function1;  
element.onclick = function2;  
etc.
```

Nota : Si se le solicita que admita navegadores anteriores a Internet Explorer 8 en su trabajo, es posible que tenga dificultades, ya que estos navegadores antiguos utilizan modelos de eventos diferentes de los navegadores más nuevos. Pero no temas, la mayoría de las bibliotecas de JavaScript (por ejemplo jQuery) tienen funciones integradas que abstraen las diferencias entre navegadores. No se preocupe demasiado por esto en esta etapa de su viaje de aprendizaje.

## Otros conceptos de eventos

En esta sección, cubriremos brevemente algunos conceptos avanzados que son relevantes para los eventos. No es importante comprenderlos completamente en este punto, pero podría servir para explicar algunos patrones de código que probablemente encontrará de vez en cuando.

## Objetos de evento

A veces dentro de una función de controlador de eventos, es posible que vea un parámetro especificado con un nombre como event, evt o simplemente e. Esto se denomina objeto de evento y se pasa automáticamente a los controladores de eventos para proporcionar características e información adicionales. Por ejemplo, reescribamos ligeramente nuestro ejemplo de color aleatorio nuevamente:

```
function bgChange(e) {  
    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';  
    e.target.style.backgroundColor = rndCol;  
}
```

```
console.log(e);  
}
```

```
btn.addEventListener('click', bgChange);
```

Nota : Puede encontrar el código fuente completo para este ejemplo en GitHub (también verlo ejecutándose en vivo ).

Aquí puede ver que estamos incluyendo un objeto de evento, `e`, en la función, y en la función configurando un estilo de color de fondo `e.target`, que es el botón en sí. La `target` propiedad del objeto de evento es siempre una referencia al elemento sobre el que acaba de ocurrir el evento. Entonces, en este ejemplo, estamos configurando un color de fondo aleatorio en el botón, no en la página.

Nota : Puede usar cualquier nombre que desee para el objeto de evento; solo necesita elegir un nombre que luego pueda usar para hacer referencia a él dentro de la función del controlador de eventos. `e/ evt/ event` se utilizan con mayor frecuencia por los desarrolladores, ya que son cortos y fáciles de recordar. Siempre es bueno ceñirse a un estándar.

`e.target` es increíblemente útil cuando desea configurar el mismo controlador de eventos en varios elementos y hacer algo con todos ellos cuando ocurre un evento en ellos. Por ejemplo, puede tener un conjunto de 16 mosaicos que desaparecen cuando se hace clic en ellos. Es útil poder configurar siempre la cosa para que desaparezca como `e.target`, en lugar de tener que seleccionarla de una manera más difícil. En el siguiente ejemplo (vea `util-eventtarget.html` para el código fuente completo; también verlo ejecutándose en vivo aquí), creamos 16 `<div>` elementos usando JavaScript. Luego, los seleccionamos todos usando `document.querySelectorAll()`, luego recorremos cada uno, agregando un `onclick` controlador a cada uno que hace que se aplique un color aleatorio a cada uno cuando se hace clic:

```
var divs = document.querySelectorAll('div');
```



```
for (var i = 0; i < divs.length; i++)  
{ divs[i].onclick = function(e)  
{ e.target.style.backgroundColor = bgChange();  
}  
}
```

El resultado es el siguiente (intente hacer clic en él, diviértase):

La mayoría de los controladores de eventos que encontrará solo tienen un conjunto estándar de propiedades y funciones (métodos) disponibles en el objeto de evento (consulte la Eventreferencia del objeto para obtener una lista completa). Sin embargo, algunos controladores más avanzados agregan propiedades especializadas que contienen datos adicionales que necesitan para funcionar. La API de Media Recorder , por ejemplo, tiene un dataavailableevento, que se activa cuando se ha grabado algún audio o video y está disponible para hacer algo (por ejemplo, guardarlo o reproducirlo). El objeto de evento del controlador de ondataavailable correspondiente tiene una datapropiedad disponible que contiene los datos de audio o video grabados para permitirle acceder a ellos y hacer algo con ellos.

### Prevenir el comportamiento predeterminado

A veces, se encontrará con una situación en la que desea evitar que un evento haga lo que hace de forma predeterminada. El ejemplo más común es el de un formulario web, por ejemplo, un

formulario de registro personalizado. Cuando completa los detalles y presiona el botón enviar, el comportamiento natural es que los datos se envíen a una página específica en el servidor para su procesamiento, y el navegador sea redirigido a una página de "mensaje de éxito" de algún tipo (o la misma página, si no se especifica otra).

El problema surge cuando el usuario no ha enviado los datos correctamente; como desarrollador, querrá detener el envío al servidor y darles un mensaje de error diciéndoles qué está mal y qué se debe hacer para corregir las cosas. Algunos navegadores admiten funciones de validación automática de datos de formularios, pero como muchos no lo hacen, se recomienda no confiar en ellas e implementar sus propias comprobaciones de validación. Veamos un ejemplo sencillo.

Primero, un formulario HTML simple que requiere que ingrese su nombre y apellido:

```
<form>
  <div>
    <label for="fname">Nombre: </label>
    <input id="fname" type="text">
  </div>
  <div>
    <label for="lname">Apellido: </label>
    <input id="lname" type="text">
  </div>
  <div>
    <input id="submit" type="submit">
  </div>
</form>
```



<p></p>

Ahora algo de JavaScript: aquí implementamos una verificación muy simple dentro de un controlador de eventos onsubmit (el evento de envío se activa en un formulario cuando se envía) que prueba si los campos de texto están vacíos. Si es así, llamamos a la preventDefault() función en el objeto de evento, que detiene el envío del formulario, y luego mostramos un mensaje de error en el párrafo debajo de nuestro formulario para decirle al usuario cuál es el problema:

```
var form = document.querySelector('form');
```

```
var fname = document.getElementById('fname'); var  
lname = document.getElementById('lname'); var  
submit = document.getElementById('submit'); var para  
= document.querySelector('p');
```

```
form.onsubmit = function(e) {
```

```
if (fname.value === "" || lname.value === "") { e.preventDefault();
```

```
    para.textContent = 'Completa ambos datos!';
```

```
    }
```

```
}
```