El futuro digital es de todos

MinTIC

UNIVERSIDAD EL BOSQUE

MisiónTIC 2022

Recurso 2.2

JavaScript
Eventos y APIs

Mag. Carlos Adolfo Beltrán Castro

Incorporando Javascript

Siguiendo los mismos lineamientos que en CSS, existen tres técnicas para incorporar código Javascript dentro de HTML. Sin embargo, al igual que en CSS, solo la inclusión de archivos externos es la recomendada a usar en HTML5.

En línea

Esta es una técnica simple para insertar Javascript en nuestro documento que se aprovecha de atributos disponibles en elementos HTML. Estos atributos son manejadores de eventos que ejecutan código de acuerdo a la acción del usuario.
Los manejadores de eventos más usados son, en general, los relacionados con el ratón, como por ejemplo onclick, onMouseOver, u onMouseOut. Sin embargo, encontraremos sitios web que implementan eventos de teclado y de la ventana, ejecutando acciones luego de que una tecla es presionada o alguna condición en la ventana del navegador cambia (por ejemplo, onload u onfocus).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
</head>
<body>
  <div id="principal">
    <p onclick="alert('hizo clic!')">Hacer Clic</p>
    <p>No puede hacer clic</p>
  </div>
</body>
</html>
```

Usando el manejador de eventos onclick en el Listado 4-1, un código es ejecutado cada vez que el usuario hace clic con el ratón sobre el texto que dice "Hacer Clic". Lo que el manejador onclick está diciendo es algo como: "cuando alguien haga clic sobre este elemento ejecute este código" y el código en este caso es una función predefinida en Javascript que muestra una pequeña ventana con el mensaje "hizo clic!".

Intente cambiar el manejador onclick por onMouseOver, por ejemplo, y verá cómo el código es ejecutado solo pasando el puntero del ratón sobre el elemento.
El uso de Javascript dentro de etiquetas HTML está permitido en HTML5, pero por las mismas razones que en CSS, esta clase de práctica no es recomendable. El código HTML se extiende innecesariamente y se hace difícil de mantener y actualizar. Así mismo, el código distribuido sobre todo el documento complica la construcción de aplicaciones útiles.
Nuevos métodos y técnicas fueron desarrollados para referenciar elementos HTML y registrar manejadores de eventos sin tener que usar código en línea (online).

Embebido

Para trabajar con códigos extensos y funciones personalizadas debemos agrupar los códigos en un mismo lugar entre etiquetas <script>. El elemento <script> actúa exactamente igual al elemento <style> usado para incorporar estilos CSS. Nos ayuda a organizar el código en un solo lugar, afectando a los elementos HTML por medio de referencias.
Del mismo modo que con el elemento <style>, en HTML5 no debemos usar ningún atributo para especificar lenguaje. Ya no es necesario incluir el atributo type en la etiqueta <script>. HTML5 asigna Javascript por defecto.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <script>
    function mostraralerta(){
      alert('hizo clic!');
    }
    function hacerclic(){
      document.getElementsByTagName('p')[0].onclick=mostraralerta;
    }
    window.onload=hacerclic;
  </script>
</head>
<body>
  <div id=”principal”>
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```

El elemento <script> y su contenido pueden ser posicionados en cualquier lugar del documento, dentro de otros elementos o entre ellos. Para mayor claridad, recomendamos siempre colocar sus códigos Javascript en la cabecera del documento y luego referenciar los elementos a ser afectados usando los métodos Javascript apropiados para ese propósito.
Actualmente existen tres métodos disponibles para referenciar elementos HTML desde Javascript:

• **getElementsByTagName** referencia un elemento por su nombre o palabra clave.
• **getElementById** referencia un elemento por el valor de su atributo id.
• **getElementsByClassName** es una nueva incorporación que nos permite referenciar un elemento por el valor de su atributo class.

Incluso si seguimos la práctica recomendada (posicionar el código dentro de la cabecera del documento), una situación debe ser considerada: el código del documento es leído de forma secuencial por el navegador y no podemos referenciar un elemento que aún no ha sido creado.

En el ejemplo anterior el código es posicionado en la cabecera del documento y es leído por el navegador previo a la creación del elemento <p> que estamos referenciando. Si hubiésemos intentado afectar el elemento <p> directamente con una referencia, hubiéramos recibido un mensaje de error anunciando que el elemento no existe. Para evitar este problema, el código fue convertido a una función llamada mostraralerta(), y la referencia al elemento <p> junto con el manejador del evento fueron colocados en una segunda función llamada hacerclic().
Las funciones son llamadas desde la última línea del código usando otro manejador de eventos (en este caso asociado con la ventana) llamado onload. Este manejador ejecutará la función hacerclic() cuando el documento sea completamente cargado y todos los elementos creados.

Es tiempo de analizar cómo el documento del Listado 4-2 es ejecutado. Primero las funciones Javascript son cargadas (declaradas) pero no ejecutadas. Luego los elementos HTML, incluidos los elementos <p>, son creados. Y finalmente, cuando el documento completo es cargado en la ventana del navegador, el evento load es disparado y la función hacerclic() es llamada.

En esta función, el método getElementsByTagName referencia todos los elementos <p>. Este método retorna un arreglo (array) conteniendo una lista de los elementos de la clase especificada encontrados en el documento. Sin embargo, usando el índice [0] al final del método indicamos que solo queremos que el primer elemento de la lista sea retornado. Una vez que este elemento es identificado, el código registra el manejador de eventos on click para el mismo. La función mostraralerta() será ejecutada cuando el evento click es disparado sobre este elemento mostrando el mensaje “hizo clic!”.

Puede parecer mucho código y trabajo para reproducir el mismo efecto logrado por una simple línea en el ejemplo del Listado 4-1. Sin embargo, considerando el potencial de HTML5 y la complejidad alcanzada por Javascript, la concentración del código en un único lugar y la apropiada organización representa una gran ventaja para nuestras futuras implementaciones y para hacer nuestros sitios web y aplicaciones fáciles de desarrollar y mantener.

Archivos externos

Los códigos Javascript crecen exponencialmente cuando agregamos nuevas funciones y aplicamos algunas de las APIs mencionadas previamente. Códigos embebidos incrementan el tamaño de nuestros documentos y los hacen repetitivos (cada documento debe volver a incluir los mismos códigos). Para reducir los tiempos de descarga, incrementar nuestra productividad y poder distribuir y reusar nuestros códigos en cada documento sin comprometer eficiencia, recomendamos grabar todos los códigos Javascript en uno o más archivos externos y llamarlos usando el atributo src:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <script src="micodigo.js"></script>
</head>
<body>
  <div id="principal">
    <p>Hacer clic</p>
    <p>No puede hacer clic</p>
  </div>
</body>
</html>
```

El elemento <script> en el ejemplo carga los códigos Javascript desde un archivo externo llamado micodigo.js. De ahora en más, podremos insertar nuestros códigos en este archivo y luego incluir el mismo en cualquier documento de nuestro sitio web que lo necesite. Desde la perspectiva del usuario esta práctica reduce tiempos de descarga y acceso a nuestro sitio web, mientras que para nosotros simplifica la organización y facilita el mantenimiento.

El elemento <script> en el ejemplo carga los códigos Javascript desde un archivo externo llamado micodigo.js. De ahora en más, podremos insertar nuestros códigos en este archivo y luego incluir el mismo en cualquier documento de nuestro sitio web que lo necesite. Desde la perspectiva del usuario, esta práctica reduce tiempos de descarga y acceso a nuestro sitio web, mientras que para nosotros simplifica la organización y facilita el mantenimiento.

Nuevos Selectores

Como vimos anteriormente, los elementos HTML tienen que ser referenciados desde Javascript para ser afectados por el código. Si recuerda de previos capítulos, CSS, y especialmente CSS3, ofrece un poderoso sistema de referencia y selección que no tiene comparación con los pocos métodos provistos por Javascript para este propósito. Los métodos getElementById, getElementsByTagName y getElementsByClassName no son suficientes para contribuir a la integración que este lenguaje necesita y sostener la relevancia que posee dentro de la especificación de HTML5. Para elevar Javascript al nivel que las circunstancias requieren, nuevas alternativas debieron ser incorporadas. Desde ahora podemos seleccionar elementos HTML aplicando toda clase de selectores CSS por medio de los nuevos métodos querySelector() y querySelectorAll().

querySelector()

Este método retorna el primer elemento que concuerda con el grupo de selectores especificados entre paréntesis. Los selectores son declarados usando comillas y la misma sintaxis CSS, como en el siguiente ejemplo:

```
function hacerclic(){
  document.querySelector("#principal p:first-
                                        child").onclick=mostraralerta;
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

En el ejemplo, el método getElementsByTagName usado anteriormente ha sido reemplazado por querySelector(). Los selectores para esta consulta en particular están referenciando al primer elemento <p> que es hijo del elemento identificado con el atributo id y el valor main.

Debido a que ya explicamos que este método solo retorna el primer elemento encontrado, probablemente notará que la pseudo clase first-child es redundante. El método querySelector() en nuestro ejemplo retornará el primer elemento <p> dentro de que es, por supuesto, su primer hijo. El propósito de este ejemplo es mostrarle que querySelector() acepta toda clase de selectores válidos CSS y ahora, del mismo modo que en CSS, Javascript también provee herramientas importantes para referenciar cada elemento en el documento.

Varios grupos de selectores pueden ser declarados separados por coma. El método querySelector() retornará el primer elemento que concuerde con cualquiera de ellos.

querySelectorAll()

En lugar de uno, el método querySelectorAll() retorna todos los elementos que concuerdan con el grupo de selectores declarados entre paréntesis. El valor retornado es un arreglo (array) conteniendo cada elemento encontrado en el orden en el que aparecen en el documento.

```
function hacerclic(){
  var lista=document.querySelectorAll("#principal p");
  lista[0].onclick=mostraralerta;
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

Note que este ejemplo no muestra el potencial de querySelectorAll(). Normalmente será utilizado para afectar a varios elementos y no solo uno, como en este caso. Para interactuar con una lista de elementos retornados por este método, podemos utilizar un bucle for:

Note que este ejemplo no muestra el potencial de querySelectorAll(). Normalmente será utilizado para afectar a varios elementos y no solo uno, como en este caso. Para interactuar con una lista de elementos retornados por este método, podemos utilizar un bucle for:

```
function hacerclic(){
  var lista=document.querySelectorAll("#principal p");
  for(var f=0; f<lista.length; f++){
    lista[f].onclick=mostraralerta;
  }
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

En este ejemplo, en lugar de seleccionar solo el primer elemento encontrado, registramos el manejador de eventos onclick para cada uno de ellos usando un bucle for. Ahora, todos los elementos <p> dentro de mostrarán una pequeña ventana cuando el usuario haga clic sobre ellos.
El método querySelectorAll(), al igual que querySelector(), puede contener uno o más grupos de selectores separados por coma. Éstos y los demás métodos estudiados pueden ser combinados para referenciar elementos a los que resulta difícil llegar. Por ejemplo, en el próximo listado, el mismo resultado del código del Listado 4-6 es logrado utilizando querySelectorAll() y getElementById() juntos:

```
function hacerclic(){
  var lista=document.getElementById('principal').
                                              querySelectorAll("p");
  lista[0].onclick=mostraralerta;
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

Usando esta técnica podemos ver qué precisos pueden ser estos métodos. Podemos combinarlos en una misma línea y luego realizar una segunda selección con otro método para alcanzar elementos dentro de los primeros. En próximos capítulos estudiaremos algunos ejemplos más.

Manejadores de eventos

Como comentamos anteriormente, el código Javascript es normalmente ejecutado luego de que el usuario realiza alguna acción. Estas acciones y otros eventos son procesados por manejadores de eventos y funciones Javascript asociadas con ellos.
Existen tres diferentes formas de registrar un evento para un elemento HTML: podemos agregar un nuevo atributo al elemento, registrar un manejador de evento como una propiedad del elemento o usar el nuevo método estándar addEventListener().

Conceptos básicos: En Javascript las acciones de los usuarios son llamadas eventos. Cuando el usuario realiza una acción, como un clic del ratón o la presión de una tecla, un evento específico para cada acción y cada elemento es disparado. Además de los eventos producidos por los usuarios existen también otros eventos disparados por el sistema (por ejemplo, el evento load que se dispara cuando el documento es completamente cargado). Estos eventos son manejados por códigos o funciones. El código que responde al evento es llamado manejador. Cuando registramos un manejador lo que hacemos es definir cómo nuestra aplicación responderá a un evento en particular. Luego de la estandarización del método addEventListener(), este procedimiento es usualmente llamado “escuchar al evento”, y lo que hacemos para preparar el código que responderá a ese evento es “agregar una escucha” a un elemento en particular

Manejadores de eventos en línea

Para evitar las complicaciones de la técnica en línea (inline), debemos registrar los eventos desde el código Javascript. Usando selectores Javascript podemos referenciar el elemento HTML y asignarle el manejador de eventos que queremos como si fuese una propiedad.

Conceptos Básicos: Los nombres de los manejadores de eventos son construidos agregando el prefijo on al nombre del evento. Por ejemplo, el nombre del manejador de eventos para el evento click es onclick. Cuando estamos hablando sobre onclick usualmente hacemos referencia al código que será ejecutado cuando el evento click se produzca.

Antes de HTML5, esta era la única técnica disponible para usar manejadores de eventos desde Javascript que funcionaba en todos los navegadores. Algunos fabricantes de navegadores estaban desarrollando sus propios sistemas, pero nada fue adoptado por todos hasta que el nuevo estándar fue declarado. Por este motivo recomendamos utilizar esta técnica por razones de compatibilidad, pero no la sugerimos para sus aplicaciones HTML5.

El método addEventListener()

El método addEventListener() es la técnica ideal y la que es considerada como estándar por la especificación de HTML5. Este método tiene tres argumentos: el nombre del evento, la función a ser ejecutada y un valor booleano (falso o verdadero) que indica cómo un evento será disparado en elementos superpuestos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <script>
    function mostraralerta(){
      alert('hizo clic!');
    }
    function hacerclic(){
      var elemento=document.getElementsByTagName('p')[0];
      elemento.addEventListener('click', mostraralerta, false);
    }
    window.addEventListener('load', hacerclic, false);
  </script>
</head>
<body>
  <div id="principal">
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```

Para organizar el código en la función hacerclic(), asignamos la referencia al elemento a una variable llamada elemento y luego agregamos la escucha para el evento click usando esa variable.

La sintaxis del método addEventListener() es la mostrada en el Listado 4-8. El primer atributo es el nombre del evento. El segundo es la función a ser ejecutada, la cual puede ser una referencia a una función (como en este caso) o una función anónima. El tercer atributo especificará, usando true (verdadero) o false (falso), cómo múltiples eventos serán disparados. Por ejemplo, si estamos escuchando al evento click en dos elementos que se encuentran anidados (uno dentro de otro), cuando el usuario hace clic sobre estos elementos dos eventos click son disparados en un orden que depende de este valor. Si el atributo es declarado como true para uno de los elementos, entonces ese evento será considerado primero y el otro luego. Normalmente el valor false es el más adecuado para la mayoría de las situaciones.

Conceptos básicos: Funciones anónimas son funciones dinámicamente declaradas y que no tienen nombre (por esto son llamadas “anónimas”). Esta clase de funciones son extremadamente útiles en Javascript, nos ayudan a organizar el código y no sobre poblar el objeto global con funciones independientes. Usaremos funciones anónimas con frecuencia en los siguientes capítulos.

Incluso cuando los resultados de aplicar esta técnica y la anterior son similares, addEventListener() nos permite agregar tantas escuchas como necesitemos para el mismo elemento. Esta distinción le otorga a addEventListener() una ventaja sobre el resto, convirtiéndola en la técnica ideal para aplicaciones HTML5.

Debido a que los eventos son la clave para sitios webs y aplicaciones interactivas, varios fueron agregados en la especificación de HTML5. En próximos capítulos estudiaremos cada uno de estos nuevos eventos y el entorno en el cual trabajan.

APIs

Javascript es tan poderoso como cualquier otro lenguaje de desarrollo en este momento. Y por la misma razón que lenguajes de programación profesionales poseen librerías para crear elementos gráficos, motores 3D para video juegos o interfaces para acceder a bases de datos, Javascript cuenta con APIs para ayudar a los programadores a lidiar con actividades complejas.

HTML5 introduce varias APIs (interfaces de programación de aplicaciones) para proveer acceso a poderosas librerías desde simple código Javascript. El potencial de estas incorporaciones es tan importante que pronto se convertirán en nuestro objeto de estudio. Veamos rápidamente sus características para obtener una perspectiva de lo que nos encontraremos en el resto del libro.
La siguiente es solo una introducción, más adelante estudiaremos cada una de estas tecnologías con mayor profundidad.

Canvas

Canvas es una API gráfica que provee una básica pero poderosa superficie de dibujo. Esta es la más maravillosa y prometedora API de todas. La posibilidad de generar e imprimir gráficos en pantalla, crear animaciones o manipular imágenes y videos (combinado con la funcionalidad restante de HTML5) abre las puertas para lo que nos podamos imaginar.
Canvas genera una imagen con pixeles que son creados y manipulados por funciones y métodos provistos específicamente para este propósito.

Drag and Drop

Drag and Drop incorpora la posibilidad de arrastrar y soltar elementos en la pantalla como lo haríamos comúnmente en aplicaciones de escritorio. Ahora, con unas pocas líneas de código, podemos hacer que un elemento esté disponible para ser arrastrado y soltado dentro de otro elemento en la pantalla. Estos elementos pueden incluir no solo gráficos sino además textos, enlaces, archivos o datos en general.

Geolocation

Geolocation es utilizada para establecer la ubicación física del dispositivo usado para acceder a la aplicación. Existen varios métodos para acceder a esta información, desde señales de red hasta el Sistema de Posicionamiento Global (GPS). Los valores retornados incluyen latitud y longitud, posibilitando la integración de esta API con otras como Google Maps, por ejemplo, o acceder a información de localización específica para la construcción de aplicaciones prácticas que trabajen en tiempo real.

Storage

Dos APIs fueron creadas con propósitos de almacenamiento de datos: Web Storage e Indexed Database. Básicamente, estas APIs transfieren la responsabilidad por el almacenamiento de datos del servidor al ordenador del usuario, pero en el caso de Web Storage y su atributo sessionStorage, esta incorporación también incrementa el nivel de control y la eficiencia de las aplicaciones web.
Web Storage contiene dos importantes atributos que son a veces considerados APIs por sí mismos: sessionStorage y localStorage.

El atributo sessionStorage es responsable por mantener consistencia sobre la duración de la sesión de una página web y preservar información temporal como el contenido de un carro de compras, asegurando los datos en caso de accidente o mal uso (cuando la aplicación es abierta en una segunda ventana, por ejemplo).

Por el otro lado, el atributo localStorage nos permite grabar contenidos extensos de información en el ordenador del usuario. La información almacenada es persistente y no expira, excepto por razones de seguridad.

Ambos atributos, sessionStorage y localStorage reemplazan la anterior función del sistema de cookies y fueron creados para superar sus limitaciones.

La segunda API, agrupada dentro de las APIs de almacenamiento pero independiente del resto, es Indexed Database. La función elemental de un sistema de base de datos es la de almacenar información indexada. Web Storage API trabaja sobre el almacenamiento de grandes o pequeñas cantidades de información, datos temporales o permanentes, pero no datos estructurados. Esta es una posibilidad solo disponible para sistemas de base de datos y la razón de la existencia de esta API.

Indexed Database es una sustitución de la API Web SQL Database. Debido a desacuerdos acerca del estándar apropiado a utilizar, ninguna de estas dos APIs ha sido completamente adoptada. De hecho, en este mismo momento, el desarrollo de Web SQL Database API (la cual había sido recibida con brazos abiertos al comienzo), ha sido cancelado.

Debido a que la API Indexed Database, también conocida como IndexedDB, luce más prometedora y tiene el apoyo de Microsoft, los desarrolladores de Firefox y Google, será nuestra opción para este libro. Sin embargo, tenga siempre presente que en este momento nuevas implementaciones de SQL están siendo consideradas y la situación podría cambiar en el futuro cercano.

File

Bajo el título de File, HTML5 ofrece varias APIs destinadas a operar con archivos. En este momento existen tres disponibles: File, File: Directories & System, y File: Writer.

Gracias a este grupo de APIs, ahora podemos crear y procesar archivos en el ordenador del usuario.

Communication

Algunas API tienen un denominador común que nos permite agruparlas juntas. Este es el caso para XMLHttpRequest Level 2, Cross Document Messaging, y Web Sockets.
Internet ha estado siempre relacionado con comunicaciones, por supuesto, pero algunos asuntos no resueltos hacían el proceso complicado y en ocasiones imposible. Tres problemas específicos fueron abordados en HTML5: la API utilizada para la creación de aplicaciones Ajax no estaba completa y era complicada de implementar a través de distintos navegadores, la comunicación entre aplicaciones no relacionadas era no existía, y no había forma de establecer una comunicación bidireccional efectiva para acceder a información en el servidor en tiempo real.

El primer problema fue resuelto con el desarrollo de XMLHttpRequest Level 2. XMLHttpRequest fue la API usada por mucho tiempo para crear aplicaciones Ajax, códigos que acceden al servidor sin recargar la página web. El nivel 2 de esta API incorpora nuevos eventos, provee más funcionalidad (con eventos que permiten hacer un seguimiento del proceso), portabilidad (la API es ahora estándar), y accesibilidad (usando solicitudes cruzadas, desde un dominio a otro).

La solución para el segundo problema fue la creación de Cross Document Messaging. Esta API ayuda a los desarrolladores a superar las limitaciones existentes para comunicar diferentes cuadros y ventanas entre sí. Ahora una comunicación segura a través de diferentes aplicaciones es ofrecida por esta API utilizando mensajes.
La solución para el último de los problemas listados anteriormente es Web Sockets. Su propósito es proveer las herramientas necesarias para la creación de aplicaciones de red que trabajan en tiempo real (por ejemplo, salas de chat). La API les permite a las aplicaciones obtener y enviar información al servidor en períodos cortos de tiempo, volviendo posible las aplicaciones en tiempo real para la web.

History

Ajax cambió la forma en la que los usuarios interactúan con sitios y aplicaciones web. Y los navegadores no estaban preparados para esta situación. History fue implementada para adaptar las aplicaciones modernas a la forma en que los navegadores hacen seguimiento de la actividad del usuario. Esta API incorpora técnicas para generar artificialmente URLs por cada paso en el proceso, ofreciendo la posibilidad de retorna a estados previos de la aplicación utilizando procedimientos estándar de navegación.

Web Workers

Esta es una API única que expande Javascript a un nuevo nivel. Este lenguaje no es un lenguaje multitarea, lo que significa que solo puede hacerse cargo de una sola tarea a la vez. Web Workers provee la posibilidad de procesar código detrás de escena (ejecutado aparte del resto), sin interferir con la actividad en la página web y del código principal. Gracias a esta API Javascript ahora puede ejecutar múltiples tareas al mismo tiempo.

Offline

Incluso hoy día, con acceso a Internet en cada lugar que vamos, quedar desconectado es aún posible. Dispositivos portátiles se encuentran en todas partes, pero no la señal para establecer comunicación. Y los ordenadores de escritorio también pueden dejarnos desconectados en los momentos más críticos. Con la combinación de atributos HTML, eventos controlados por Javascript y archivos de texto, Offline permitirá a las aplicaciones trabajar en línea o desconectados, de acuerdo a la situación del usuario.