Programación con Javascript

Unidad 6: Funciones y Eventos

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 6: Funciones y Eventos

Objetivos

Que el alumno logre:

- Crear y aplicar funciones desarrolladas con Javascript

Eventos

Hasta ahora, todas las aplicaciones y scripts que hemos 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 algo limitadas, 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 mouse, 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 mouse, 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".

Para referirnos a un evento en HTML, el nombre del evento irá precedido por "on". Por ejemplo, el gestor de eventos de "Click" será "onClick". La forma de llamar a un evento es la siguiente.

```
<form ....>
<input type="button" onClick="función([parámetros])">
</form ....>
```

Con esta acción, asociamos al evento click sobre el botón las acciones que realice la función.

Eventos en Javascript

DragDrop: arrastrar un objeto a la ventana del navegador.

Error: se produce un error en la carga de un documento.

Focus: el usuario se posiciona en una ventana o cuadro de texto de un formulario.

KeyDown: se pulsa una tecla.

KeyPress: se pulsa o libera una tecla

KeyUp: se libera una tecla

Load: se carga un documento en el navegador

MouseDown: se pulsa un botón del ratón

MouseMove: se mueve el cursor

MouseOver: el puntero del ratón se posiciona sobre un enlace

MouseOut: el puntero del ratón sale de un enlace o imagen mapa

MouseUp: se libera un botón del ratón.

Move: se mueve la ventana. Esta acción también la puede realizar el script.

Reset: se pulsa sobre el botón reset del formulario.

Resize: las dimensiones de la ventana cambian.

Select: se selecciona una de las opciones de un cuadro combo del formulario.

Submit: se pulsa el botón submit del formulario.

UnLoad: el usuario sale de la página.

Debemos recordar que para llamar a los eventos, se debe anteponer "on" al nombre del evento.

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 navegado	<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 **onload** para esperar a que se cargue la página por completo, los eventos **onclick**, **onmouseover**, **onmouseout** para controlar el ratón y **onsubmit** para controlar el envío de los formularios.

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.

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

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 haga clic con el mouse sobre un botón:

```
<input type="button" value="Hacer clic acá" onclick="alert('Gracias por presionar');" />
```

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 hacer clic con el mouse, cuyo nombre es onclick. Así, el elemento HTML para el que se quiere definir este evento, debe incluir un atributo llamado onclick.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (alert('Gracias por presionar');), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario hace clic sobre el elemento se muestra un mensaje y cuando el usuario pasa el mouse por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Bienvenido');" onmouseover="alert('Gracias por visitarnos!');"> Puedes hacer clic sobre este elemento o simplemente pasar el mouse por encima </div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');"> ... </body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

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 mouse por encima del , el color del borde se muestra de color negro. Cuando el mouse 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:1px 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:1px 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:1px solid silver"
onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">  Sección de contenidos... </div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo id del .

Manejadores de eventos como funciones externas

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 hacer clic sobre un botón:

```
<input type="button" value="Bienvenido" onclick="alert('Gracias por visitarnos');" />
Utilizando funciones externas se puede transformar en: function muestraMensaje()
{  alert('Gracias por visitarnos'); }
<input type="button" value="Bienvenido" 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:

```
function resalta(elemento) {
switch(elemento.style.borderColor) {
case 'silver':
elemento.style.borderColor = 'black';
break;
case 'black':
elemento.style.borderColor = 'silver';
break;
}
}
<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 hemos 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. Por eso solo vamos a utilizarlo en aplicaciones sencillas.

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:

```
<input id="clickeable" type="button" value="Bienvenido" onclick="alert('Gracias por visitarnos');" />
```

Se puede transformar en:

```
// Función externa
function muestraMensaje() {
alert('Gracias por pinchar');
}
// Asignar la función externa al elemento
document.getElementById("clickeable").onclick = muestraMensaje;

// Elemento HTML
<input id="clickeable" type="button" value="Bienvenido" />
```

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.

El último paso es la clave de esta técnica.

En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("clickeable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar.

En este caso, la propiedad es onclick:

```
document.getElementById("clickeable").onclick = ...
```

Por último, se 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("clickeable").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 del elemento.

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

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos HTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

```
window.onload = function() {
document.getElementById("clickeable").onclick = muestraMensaje;
}
```

La técnica anterior utiliza el concepto de funciones anónimas, que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {   ... }
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo input=text de un formulario complejo:

```
function resalta() {
// Código JavaScript
}
window.onload = function() {
var formulario = document.getElementById("formulario");
var camposInput = formulario.getElementsByTagName("input");     f
```

```
or(var i=0; i<camposInput.length; i++) {
if(camposInput[i].type == "text") {
camposInput[i].onclick = resalta;
  }
 }
}
```

Resumen

En esta Unidad…

Trabajamos con Funciones y Eventos

En la próxima Unidad…

Trabajaremos con formularios y validación