Manipulando el DOM con JavaScript

Vicent Moncho Mas
Gemma Subirana Grau

PID_00220488

Universitat Oberta de Catalunya

Índice

1. Manipulación de la página web

En este apartado se va a presentar un conjunto de ejemplos de código JavaScript que, basándose en las técnicas estudiadas en los módulos anteriores, proporcionará una visión de las posibilidades que ofrece el lenguaje al programador.

El apartado se estructura de la siguiente manera:

- Las características del cliente.
- Las ventanas.
- Los marcos.
- La navegación y el paso de variables.

A lo largo del apartado se plantearán ejemplos sencillos que indican la base de las técnicas que permiten un control necesario en la creación de webs de cierta complejidad.

1.1. Las características del cliente

Aunque el estándar DOM de la W3C se va implementando en los principales navegadores paso a paso, todavía no se está en una situación de convergencia o compatibilidad real entre el conjunto de navegadores.

El hecho anterior obliga a que el programador web sea consciente de que algún código puede funcionar de manera distinta o simplemente no funcionar en cierta versión del navegador de un fabricante determinado; por ello, es muy importante que se pueda conocer, antes de ejecutar un script, cuál es el cliente que lo interpretará.

1.1.1. Consultar el navegador

Es posible consultar el fabricante del navegador a partir de la propiedad "appName" del objeto Navigator. El retorno de este método es el siguiente:

- "Netscape" cuando el cliente es Firefox, Safari o Chrome.
- "Microsoft Internet Explorer" cuando el cliente es Internet Explorer.

En el caso de Opera, la función puede devolver cualquiera de las opciones anteriores más "Opera", ya que depende de ciertos parámetros de configuración del navegador. Por lo tanto, se deberá consultar la propiedad "userAgent", que proporciona más información.

Pero la información más completa sobre la versión de un navegador se obtiene a partir del método userAgent del objeto Navigator. Este método devuelve una cadena que contiene un conjunto de parámetros que, además de identificar al fabricante, identificará la versión concreta. Para ello, será necesario realizar cierta búsqueda en la cadena de información clave, por ejemplo:

- La subcadena MSIE identifica a Internet Explorer y va seguida de un espacio y el número de versión del navegador.
- La subcadena Gecko identifica al motor implementado en el navegador. Este motor es utilizado en los navegadores Mozilla, Netscape, Firefox y Camino.
- Si aparece la subcadena Safari y no aparece Chrome, se está indicando que el motor implementado en el navegador es el de Safari.
- La subcadena Opera identifica que el navegador es Opera.
- La subcadena Chrome indica que el navegador es Chrome.

Por lo tanto, la implementación de esta búsqueda se puede realizar con la siguiente sintaxis:

```
var agent = navigator.userAgent;
if (agent.indexOf("subcadena") != -1)
```

donde "subcadena" debe ser substituido por la subcadena del navegador que se quiera consultar.

1.1.2. Consultar el sistema operativo

El mismo código anterior puede servir para detectar el sistema operativo sobre el que se ejecuta el navegador, ya que éste se identifica en la cadena que devuelve el método userAgent.

La subcadena que se debe buscar en cada uno de los casos sería:

- "Win" en el caso de entornos Microsoft.
- "Mac" en el caso de entornos Macintosh.
- "X11" en el caso de entornos basados en UNIX.

1.1.3. Consultar si el navegador soporta W3C DOM

El método hasFeature() del objeto Implementation se utiliza para consultar desde un script si el navegador soporta uno de los módulos del estándar del W3C. En el siguiente ejemplo se carga una hoja de estilos u otra, dependiendo de si el navegador soporta el estándar CSS del DOM2.

Web recomendada

Podéis consultar la lista de todos lo módulos del nivel 2 de W3C.

```
var ficheroCSS;
if (document.implementation.hasFeature("CSS2","2.0")){
   ficheroCSS = "estiloCSS2.css";
} else {
   ficheroCSS = "estiloCSS1.css";
}
document.write(" ");("<link rel='stylesheet' type='text/css' href='"+ficheroCSS +"'>");
```

1.2. Las ventanas

El manejo de las ventanas es un aspecto controvertido en la programación con JavaScript. Desde el inicio, las ventanas han formado parte de un conjunto de recursos de los programadores de JavaScript para enriquecer la usabilidad o para entorpecer y atormentar al usuario con una amalgama de ventanas sin sentido.

El objeto Window no es independiente de la divergencia de los modelos de navegador; las propiedades y métodos que se implementan dependen en gran medida del navegador.

El abuso de las ventanas emergentes ha llegado a tal punto que los navegadores actuales disponen de una utilidad para bloquearlas configurable por el usuario. Esto ya indica que de cara a realizar una programación con un cierto nivel de usabilidad, se debe intentar evitar el uso de ventanas emergentes e incluir en la ventana actual todo el contenido.

Con el objetivo de establecer mecanismos sólidos de seguridad en la programación con JavaScript, existen una serie de restricciones sobre lo que se puede hacer con las ventanas. A continuación, se enumeran algunas de las acciones que no se pueden realizar con JavaScript:

- Modificar el marco de la ventana actual; aunque es posible redimensionar y situar la ventana del navegador, no se pueden agregar o eliminar los componentes que forman el marco (barra de menú, estado, scroll, etc.). Esto sólo es posible realizarlo en ventanas que son creadas desde un script.

- Cerrar la ventana principal desde un script que se ejecuta en una subventana; en el caso de que se realice la llamada, aparecerá un cuadro de diálogo que solicita permiso al usuario para realizar el cierre.

- Cerrar ventanas no creadas por un script. De hecho, no es posible acceder a ventanas no creadas desde un script. No existe un array con las ventanas existentes.

- Acceder a propiedades de Document desde otras ventanas que proceden de otros dominios, es decir, no es posible acceder a datos de una página HTML desde otra si éstos no proceden del mismo dominio.

- Capturar la pulsación de los botones del navegador. Esto permitiría poder controlar desde un script el comportamiento del navegador y sería realmente peligroso.

- Cambiar la dirección de la barra de direcciones; sólo es posible cargar una nueva página, pero no con una página cargada modificar el URL de ésta (de algún modo, se estaría suplantando su identidad).

- Modificar elementos de la lista de favoritos.

Si se analizan las limitaciones anteriores en profundidad, se llega fácilmente a la conclusión de que son necesarias de cara a disponer de un entorno seguro y fiable.

Revisadas las limitaciones, a continuación se van a presentar ciertas acciones que se pueden realizar con las ventanas y que tienen su utilidad.

1.2.1. Posicionar y maximizar la ventana principal

Situar la ventana principal del navegador en un punto específico de la pantalla es un proceso muy simple. Para ello, tenemos el método moveTo(x,y), que posiciona la ventana en las coordenadas en píxeles que se le pasan como parámetros.

En el caso de que no se quiera situar la ventana en una posición de la pantalla, sino realizar un cierto desplazamiento sobre la posición actual, se debe utilizar el método moveBy(x,y).

La implementación de los anteriores movimientos no provoca ningún cambio en el tamaño de la ventana. Para modificar el tamaño de la ventana, se puede utilizar el método resizeTo(x,y), cuyos parámetros son los píxeles que indican el tamaño que debe tener ésta.

Si se quiere maximizar una ventana del navegador, se puede utilizar el siguiente código:

```
<html>
<head>
<script type="text/javascript">
function abraF()
{
ventana=window.open('','','width=200,height=100');
ventana.document.write("<p>Mi ventana </p>");
```

```
}

function maximiza()
{
ventana.moveTo(0,0);
ventana.resizeTo(screen.availWidth,screen.availHeight);
ventana.focus();
}
</script>
</head>
<body>

<input type="button" value="Crea la ventana " onclick="abraF()" />
<br /><br />
<input type="button" value="Maximiza ventana " onclick="maximiza()" />

</body>
</html>
```

De la función anterior se deben tener en cuenta los siguientes detalles:

- En primer lugar, se sitúa la ventana en la parte superior izquierda con la sentencia moveTo(0,0).

- En segundo lugar, se cambia el tamaño al valor máximo disponible de anchura y altura. Estos dos valores se consiguen a partir de las propiedades availWidth y availHeight del objeto Screen.

1.2.2. Crear una nueva ventana y asignarle el foco

La creación de una nueva ventana se implementa con el método open del objeto Window:

```
var nuevaVentana = window.open("Pagina1.html", "Nueva página", "status,menubar,
 height=400,width=300");
```

En el ejemplo de código anterior, en la nueva ventana se va a cargar la página "Pagina1.html" y su título será "Nueva página". Las características de la página se especifican en la cadena del tercer parámetro.

Ved también

Podéis ver la relación completa de las características de la página en el apartado 3 del módulo "Introducción al DOM" de esta asignatura.

En el caso de que se quiera crear la página desde cero, se debe dejar el primer parámetro vacío, de modo que, a continuación, se irá creando el contenido de la página utilizando los métodos write y writeln del objeto Document.

Cuando el usuario está trabajando y existen varias ventanas, en realidad sólo está trabajando con la página que tiene el foco activo o, por ejemplificarlo de forma visual, la que está en primer plano. El método que permite subir una ventana que se encuentra en un segundo plano (escondida detrás del resto) al primer plano es focus().

El siguiente código buscará si una ventana está abierta; si no lo está, la crea, y si lo está, la trae al primer plano:

```
var nuevaVentana;
function traeLaVentana(url){
   if (!nuevaVentana || nuevaVentana.closed) {
      nuevaVentana = window.open(url, "Nueva", "status, height=200, width=300");
   } else {
      nuevaVentana.focus();
   }
}
```

1.2.3. Comunicación entre ventanas

En el subapartado anterior, se ha visto cómo se puede crear una nueva ventana desde una ventana principal. En éste veremos cómo se puede comunicar ventanas desde JavaScript que tienen una relación de "parentesco" y que, tal como se comentaba en la introducción, provienen de un mismo dominio.

En el siguiente ejemplo, se va a crear una nueva ventana sobre la que se escribirá el contenido, y todo esto desde la ventana padre:

```
<html>
<script type="text/JavaScript">
//Se define una variable global que almacenará una referencia a la ventana
var nuevaVentana;
//La siguiente función genera y llena de contenido la nueva ventana
function creaVentana(){
   //Comprobamos que no esté creada
   if (!nuevaVentana || nuevaVentana.closed) {
      nuevaVentana = window.open("","Nueva", "status, height=200, width=300");
      //Retrasamos la escritura 50 ms para evitar problemas
      setTimeout("pintaVentana()",50);
   } else if (nuevaVentana.focus) {
      //La ventana está abierta y es accesible; se trae al primer plano
   nuevaVentana.focus();
   }
}
function pintaVentana(){
   //Montamos el contenido de la nueva ventana
   var contenido = "<html><head><title>Segunda ventana</title></head>";
```

```
    contenido += "<body><h1>Ventana creada desde un script</h1>";
    contenido += "</body></html>";
    //Se escribe el HTML en el nuevo documento de la ventana
    nuevaVentana.document.write(contenido);
    nuevaVentana.document.close();
}
</script>
</head>
<body>
<form>
<input type="button" value="Crea ventana" id= "BT1" onclick="creaVentana();"/>
</form>
</body>
</html>
```

Respecto al código anterior, sólo cabe comentar que la llamada a la escritura del contenido a la ventana recién creada se realiza utilizando el temporalizador, ya que se puede dar el caso de que no se haya finalizado la creación de la pantalla y ya se esté intentando pintar sobre ella (este efecto puede darse debido a la ejecución en paralelo del código que se da en los nuevos navegadores).

Otro aspecto que se debe tener en cuenta es que la variable *nuevaVentana* almacena una referencia a la ventana recién creada y es el mecanismo de acceso a la nueva ventana desde la principal.

Posiblemente, el problema fundamental sería cómo acceder en sentido contrario, es decir, desde la ventana recién creada hacer referencia a un valor, propiedad o método de la ventana creadora o padre de ésta.

Esto se puede implementar gracias a la propiedad opener de Window, ya que referencia a la ventana que ha sido la responsable de la ejecución de la sentencia window.open. De este modo, siguiendo el ejemplo anterior desde la ventana secundaria recién creada se podría acceder al valor del botón del formulario con la siguiente sintaxis:

```
window.opener.document.forms[0].BT1.value
```

El valor de la propiedad opener será "null" en el caso de que la ventana no haya sido creada desde otra ventana. Esta propiedad permite comprobar si la ventana actual ha sido creada por código o por el usuario:

```
if (typeof window.opener == "object") {
    //La ventana fue abierta por un script
}
```

1.3. Los marcos

Igual que ocurre con las ventanas modales, el uso de marcos también ha generado controversia por parte de los programadores. Las características de los marcos los hacen útiles en algunos casos específicos. La aplicación más común es la de dividir la página en un marco de contenido y un pequeño marco en el que se situará el índice. Con esta estructura, el usuario navega por el marco de contenido guardando el marco de índice de manera estable (incluso se puede utilizar como contenedor de ciertas variables que deben permanecer).

Los marcos establecen entre sí un conjunto de relaciones jerárquicas igual que los objetos del DOM. El documento inicial que carga el navegador contiene el elemento frameset (que define la estructura de marcos de la ventana y es el padre de todos los marcos que se crean).

Los marcos son, a su vez, un objeto Window y, por lo tanto, se pueden utilizar las propiedades y métodos del objeto Window. Respecto al acceso a los marcos, éste se puede implementar a partir del array frames[ ] del objeto Window, que contiene una referencia a cada uno de los marcos que se han definido.

La estructura jerárquica puede tener más de un nivel, es decir, en un marco se puede crear un frameset en el que se defina una nueva estructura de marcos, con los que aparecen relaciones de segundo nivel entre el objeto inicial y los marcos finales. Cuando se tiene este tipo de relaciones, el acceso se puede efectuar de la siguiente manera:

- Encadenando las llamadas window.frames[1].frames[0] se accede a un marco situado en dos niveles en la jerarquía.

- Con la propiedad parent se accede al contenedor superior del marco que realiza la llamada: parent.frames[1], que hace referencia al segundo frame de la estructura.

- Con la propiedad top se accede al contenedor "supremo", es decir, aquél que está en lo más alto de la jerarquía.

Con los mecanismos de acceso anteriores y con el hecho de que un marco es en sí mismo una ventana, parece posible realizar cualquier acción desde un marco sobre su propia estructura, pero no es así. Existe un conjunto pequeño de restricciones que, como es de esperar, está relacionado con la seguridad:

- No es posible acceder a las propiedades de un documento desde otro marco si las páginas pertenecen a dominios distintos.

- No es posible cambiar el URL de la barra de direcciones; ésta mostrará la dirección del documento que contiene la estructura de los marcos y no la dirección de la página de uno de los marcos.

- No es posible introducir en el array de Favoritos el URL de uno de los documentos cargado en uno de los marcos; el URL será el de la página que contiene la estructura de marcos.

A continuación, se van a presentar ciertas técnicas que son útiles cuando se trabaja con marcos en JavaScript.

1.3.1. Crear un marco vacío

Es posible crear un marco que no tenga contenido sin la necesidad de crear un documento HTML en blanco que se cargue en el marco en cuestión. El mecanismo es muy simple, tal como se aprecia en el ejemplo:

```
<html>
<head>
<script type="text/JavaScript">
function marcoVacio(){
   return "<html><body></body></html>";
}
</script>
</head>
<frameset rows="50,*">
   <frame name="frame1" id="frame1" src="menu.html">
   <frame name="frame2" id="frame2"
      src="JavaScript:parent.marcoVacio()">
</frameset>
</html>
```

Realmente, la técnica consiste en sustituir la dirección del fichero HTML por una función que está creando una página HTML en blanco. Es más, se podría crear la página de manera dinámica desde la propia función marcoVacio() con cierto contenido.

1.3.2. Asegurar que una página carga su estructura de marcos

Uno de los inconvenientes como consecuencia del uso de marcos se produce cuando en una estructura de marcos, en un marco en concreto, se carga de nuevo otra estructura de marcos:

```
if (top.location.href == self.location.href) {
   top.location.href ="conjuntoMarcos.html";}
```

El código compara los URL de las ventanas referidas como top y self; si son la misma, significa que el documento es el único que se carga en la ventana del navegador y, por lo tanto, la estructura de frames ocupa la página inicial del navegador y no está cargada en un marco.

Tal como se observa, en el código anterior "conjuntoMarcos.html" es la página que contiene la estructura de marcos y el nombre del fichero deberá cambiarse en cada script.

1.3.3. Cambiar el contenido de un marco desde otro

Desde un marco es posible cambiar el contenido de otro marco a través del cambio del URL que indica la página web que está cargada en el marco. Para ello, se puede asignar el URL a la propiedad location.href del marco hijo con la siguiente sintaxis:

```
parent.otroFrame.location.href = "nuevaPagina";
```

Donde "otroFrame" es el nombre o id del marco al que se le quiere modificar el contenido.

Es decir, si no se quisiera modificar el marco mediante la carga de una nueva página HTML, sino a partir de la escritura desde el mismo script. Esto es posible por el hecho de que un marco es en sí mismo una ventana window:

```
parent.otroFrame.document.write = "Introducimos aquí el código HTML que se quiera";
```

En el caso de que se necesite cambiar simultáneamente el contenido de dos o más marcos, se puede implementar de manera sencilla a partir de una función:

```
function cargaMarcos(url1, url2) {
   parent.otroFrame1.location.href = url1;
   parent.otroFrame2.location.href = url2;
   return false;
}
```

Por lo tanto, se puede llamar a la anterior función con los dos parámetros que indican las páginas web que se cargarán en cada uno de los marcos. La función puede ser variada a gusto del programador para introducir más parámetros (se podrían pasar como parámetro los nombres de los frames) e incluso se podrían modificar para que cargaran el contenido HTML a partir de código JavaScript.

1.3.4. Sustituir una estructura de marcos por una página

En ocasiones, hay cargada una estructura de marcos en una página web y se quiere sustituir esta estructura por una página simple, eliminando la estructura de marcos. Para realizar esta sustitución, se debe asegurar que la asignación del URL se realiza en la ventana del navegador que contiene la estructura de marcos inicial. Para ello, se puede utilizar la propiedad top del objeto Window:

```
top.location.href = "nuevaPagina.html"
```

En el caso de que se tratara de una estructura de marcos sencilla, se podría utilizar parent en lugar de top, pero con este último se asegura que se está accediendo a la ventana padre de todas.

1.3.5. Evitar que una página sea cargada en un marco

En ocasiones, se observa cómo un sitio web se carga en un marco de otro sitio web y, por lo tanto, aparece desconfigurado, con scrolls laterales, por lo que resulta bastante incomodo.

El anterior efecto es evitable a partir de la inserción de código JavaScript en el head de la página de inicio, que comprueba que no se esté cargando la página en una estructura de marcos. El código es el siguiente:

```
if (top != self) {
   top.location.href = location.href;
}
```

De hecho, el anterior código puede utilizarse para evitar que una estructura de marcos se cargue en un marco de otra estructura de marcos. Para ello, el código anterior sólo debería cargarse en la página que define la estructura de marcos.

1.4. Navegación y menús

En este subapartado, se va a tener en cuenta una serie de características especiales que tiene la web, en particular:

a) Se estudiarán algunas **utilidades del objeto Location**, que permitirán controlar algunos aspectos sobre la navegación. En este sentido, las políticas de seguridad implantadas dejan poco margen de maniobra.

b) Se estudiarán varias **técnicas que permiten el paso de información entre distintas páginas web**; en particular, se trata de cómo pasar datos a partir del uso de cookies, marcos y mediante el URL.

c) Se estudiarán las **técnicas que permiten la creación de menús de navegación más complejos** en la página web.

1.4.1. Cargar una nueva página o un enlace interno

Desde el script, se quiere forzar la carga de una nueva página web, es decir, se quiere forzar la navegación al siguiente destino (seguramente como consecuencia de un evento de usuario o de navegador). Para realizar la carga de la nueva página en la ventana del navegador o marco actual, simplemente se asigna el URL a la propiedad location.href:

```
location.href = "http://www.uoc.edu";
```

En el caso de que se quiera ir a un enlace interno de la página, se puede asignar la cadena del nombre del enlace (es decir, el valor asignado al atributo name de un elemento a) o al atributo id de cualquier elemento).

```
location.hash = "fase3";
```

Si en lugar de cargar la nueva página mediante el método location.href se realiza a través del método location.replace(), se consigue que la página inicial no aparezca en el historial de navegación. Esto es debido a que el método replace no sólo modifica la página que hay en location, sino también la que aparece en el array del historial, por lo que la anterior es sustituida por la nueva y deja de aparecer.

En ocasiones, es necesario navegar a un cierto destino a partir de una elección realizada en una lista desplegable de valores. Este tipo de navegación necesita las siguientes técnicas:

- Cada opción de la lista desplegable debe almacenar en la propiedad value el destino asociado al valor.

- Se debe asociar un manejador de eventos al objeto Select que lance la función que realmente abre el nuevo destino seleccionado.

- Se debe crear la función que manejará el evento anterior y que se encargará de cargar la nueva página en el navegador.

El siguiente código es un ejemplo de la técnica planteada. En primer lugar, se muestra el código HTML de la lista desplegable:

```
<select name="destinos" id="destinos" onchange="navega(this)">
   <option value = "">Elige tu destino:</option>
   <option value="http://www.barbados.com/">Islas Barbados</option>
   <option value="http://www.formentera.es">Formentera</option>
```

```
</select>
```

Se puede observar en la primera línea cómo se ha asociado el manejador de eventos sobre el objeto Select, utilizando para ello el evento onchage (que se ejecuta cuando se ejecuta una selección) y que llama a la función navega que se presenta a continuación:

```
function navega(eleccion){
   var url = destinos.options[destinos.selectedIndex].value;
   if (url) {
      location.href = url;
   }
}
```

En el caso de que se quiera utilizar la sintaxis del DOM estándar, se puede realizar, en primer lugar, la asignación del evento al objeto Select de la siguiente manera:

Nota

Hay que tener en cuenta que esta asignación hay que hacerla después de la creación del objeto Select "destinos", ya que en caso contrario se estaría haciendo referencia a un objeto que aún no existe.

```
document.getElementById("destinos").onchange = navega;
```

y la función manejadora se puede reescribir de modo que utilice el objeto Event y que sea compatible tanto con el modelo de objetos del DOM estándar como con IE, de la siguiente manera:

```
function navega(evento){
   evento=(evento) ? evento : ((event) ? event : null);
   if (evento) {
      var elemento = (evento.target) ? evento.target : ((evento.srcElement) ?
       evento.srcElement : null);
      if (elemento && elemento.tagName.toLowerCase() == "select" && elemento.value) {
         location.href = elemento.value;
      }
   }
}
```

1.4.2. Paso de variables entre páginas

Una primera técnica que permite el paso de valores o variables entre distintas páginas utiliza una combinación entre las funciones de manejo de cookies y los eventos onunload y onload del objeto Document.

Ved también

Las funciones de manejo de cookies fueron tratadas en el apartado 4 del módulo "Orientación a objetos en JavaScript" de esta asignatura.

A continuación, se presenta un ejemplo en el que se almacena el valor de un campo de un formulario y lo guarda en una cookie durante 30 días:

```
<script type="text/JavaScript">
function guardaDatos() {
   var datos = document.forms[0].nombre.value;
```

```
    asignaCookie("Nombre",datos,30);
}
</script>
```

de manera que, al cerrar el documento, se llamará a la función guardaDatos:

```
<body onunload = "guardaDatos()">
```

De la misma manera, el documento que quiera recuperar los valores, deberá llamar en el evento onload a la función que recupera los valores de la cookie:

```
<script type = "text/JavaScript">
function recuperaValores() {
   extraeCookies();
   var nombre = cookies["Nombre"];
}
</script>
```

Si se sitúa el código anterior en el head de la página, en la línea de definición del cuerpo se debería realizar la llamada de la función recuperaValores al producirse el evento load:

```
<body onload = "recuperaValores()">
```

El principal problema que puede surgir al utilizar cookies son las posibles opciones en la configuración de seguridad del navegador del cliente.

Una alternativa al uso de cookies es el almacenamiento de los valores utilizando las características de la estructura de marcos; en particular, la estructura Window representa la estructura de marcos que permanece fija mientas los documentos van entrando y saliendo de los marcos hijo. Esta ventana superior es capaz de almacenar datos JavaScript de todo tipo en variables globales.

Se podrá utilizar el manejador de eventos unload de la página de uno de los marcos para almacenar los datos en una variable global de la ventana que contiene la estructura de los marcos.

A continuación, se plantea un ejemplo de la técnica planteada:

```
<script type = "text/JavaScript">
   function almacenaDatos() {
      top.nombre = document.forms[0].nombre.value;
   }
</script>
```

de manera que, en el cuerpo de la página que necesita almacenar los valores, se inserta el siguiente código:

```
<body onunload="almacenaDatos()">
```

En el documento que debe recuperar los valores se inserta el siguiente código:

```
<script type="text/JavaScript">
function leeValores(){
   if (typeof top.nombre != "undefined") {
      document.forms[0].nombre.value = top.nombre;
   }
}
</script>
```

y en la definición del cuerpo se inserta la referencia al manejador de eventos:

```
<body onload="leeValores()">
```

Los principales problemas de la técnica anterior se deben, en primer lugar, al rechazo en el uso de marcos por parte de los programadores y, en segundo lugar, a que esta técnica funciona mientras la página que contiene la estructura de marcos se mantenga en la página cargada; por lo tanto, si en algún momento se forzara una recarga de la página utilizando la función de recarga del navegador, el valor de la variables almacenadas se perdería.

Para finalizar este subapartado, una de las técnicas más utilizadas se basa en el paso de las variables mediante el URL que abre la nueva página. Estos datos se pasarán como una cadena de búsqueda que se añadirá al URL de la siguiente página y se incluirá un script en la página posterior que se encargará de leer la cadena de búsqueda y recuperar los datos.

A continuación, se puede ver la técnica en un ejemplo sencillo:

```
<script type="text/JavaScript">
function navega(url) {
   var datosApasar = document.forms[0].nombre.value;
   location.href = url + "?" + encodeURIComponent(datosApasar);
}
</script>
```

Por lo tanto, la función anterior abre la siguiente página en el navegador, pero añade al URL una cadena de datos. Esta función deberá ser llamada desde los enlaces que sean necesarios, por lo tanto, se deberá implementar en el evento onclick de los enlaces:

```
<a href="paginaFin.html" onclick="navega('paginaFin.html'); return false">...</a>
```

De esta manera, desde la página receptora se habrá de implementar un script que sea capaz de localizar y tratar la cadena que se ha pasado. Esta función deberá ser llamada en el evento load del objeto Document para que los valores estén disponibles en el momento de la carga de la página:

```
<script type="text/JavaScript">
function leeDatos() {
   var buscaCadena = decodeURIComponent(location.search.substring(1,
    location.search.length));
   if (buscaCadena.length > 0) {
      document.forms[0].userName.value = buscaCadena;
   }
}
</script>
```

de manera que en el evento load de Document se referenciará la función anterior:

```
<body onload="leeDatos()">
```

La principal ventaja del mecanismo anterior es que es totalmente independiente del navegador, ya que todos los navegadores respetan el paso de cadenas de búsqueda en el URL, por lo tanto, de las tres técnicas vistas respecto al paso de variables, ésta es la óptima o preferida por la mayoría de los programadores.

2. Formularios dinámicos

Los formularios en la web fueron el primer objetivo del lenguaje de programación JavaScript y actualmente siguen haciendo un uso intensivo de scripts, con el objetivo de aumentar la interacción inmediata con el usuario.

De esta manera, la validación de formularios mediante JavaScript sigue siendo útil, ya que acelera la corrección de errores, pero no implica que la validación por parte del lado del servidor no deba realizarse, ya que ésta es inevitable.

Respecto a la validación de formularios, se deben tener en cuenta dos tipos de estrategia en el proceso de validación: en tiempo real y al enviar el formulario.

a) La **validación en tiempo real** se basa en la detección de cierta actividad y la llamada a una función cuando ésta ocurre.

Validación en tiempo real

Por ejemplo, el evento onChange se produce cuando cambia el contenido de un cuadro de texto y este evento puede llamar a una función que valide el nuevo texto introducido.

La principal ventaja de la validación en tiempo real es que el usuario todavía tiene reciente la información introducida en el campo, es decir, cuando el usuario acaba de introducir el contenido de un campo, en caso de un error en el contenido, se puede avisar inmediatamente al usuario.

b) La **validación al enviar el formulario** se basa en la comprobación de cada uno de los componentes del formulario justo antes de que éste sea enviado al servidor. De esta manera, a partir del evento submit del objeto Form se lanza el script que realiza el conjunto de validaciones.

En este apartado se van a estudiar algunas técnicas que permitirán realizar las acciones básicas en el manejo de formularios; en particular, se verán técnicas relacionadas con:

- Control del cursor y del foco.
- Control de los campos del formulario.
- Validación de campos.

2.1. Control del cursor y del foco

En este subapartado se verán distintas técnicas que mejorarán la usabilidad del formulario, que, aunque puedan ser consideradas como pequeños detalles, ayudan a los usuarios de manera casi transparente.

2.1.1. Enviar el cursor a un campo determinado

Cuando se entra en la web de Google, el usuario sólo debe introducir las cadenas clave que buscar y pulsar la tecla Intro. Se trata de un proceso muy simple y es uno de los factores del éxito de Google, pero detrás de este funcionamiento existe código que hace que el campo de texto de búsqueda reciba el foco en cuanto se abra la página de inicio.

El código que envía el cursor al primer campo de texto del formulario es muy simple. En el ejemplo que se presenta a continuación, el código se enlaza al manejador del evento onload del objeto Document, ya que de esta manera, una vez cargada la página web, la primera acción que se realizará será mover el foco al campo de destino:

```
<body onload="document.formulario.campo.focus()">
```

En el caso de que el campo de texto tenga un texto predeterminado, es una ayuda que este texto aparezca inicialmente seleccionado, ya que de este modo, cuando el usuario pulse cualquier tecla, se sustituirá el contenido predeterminado por el nuevo texto (evitando el proceso de eliminación del texto predeterminado).

La selección del texto en un campo de texto se realiza con el método select(). Así pues, el código anterior quedaría de la siguiente manera:

```
<body onload="document.formulario.campo.focus();
document.formulario.campo.select()">
```

Por lo tanto, en primer lugar, se sitúa el foco y, a continuación, se selecciona el texto contenido en el campo.

Si en el proceso de validación de un campo de texto éste no la supera, normalmente se devuelve el cursor al campo erróneo seleccionando el texto erróneo, con el objetivo de que el usuario sobrescriba el texto con el valor correcto.

Los métodos que permiten estas acciones son focus() y select(), pero deben ser llamados en un cierto punto del código (desde la función de validación). Para ello, a continuación, se plantea una función que encapsula estos dos métodos:

```
function asignaFoco(nombreForm, nombreCampo) {
   var elemento = document.forms[nombreForm].elements[nombreCampo];
   elemento.focus();
   elemento.select();
}
```

Sobre la función anterior, se debe tener en cuenta que es válida para campos de tipo texto, ya que en el caso de botones de opción o listas desplegables la selección del contenido no tiene ningún sentido.

2.1.2. Avanzar el foco mediante la tecla intro

El movimiento entre los distintos campos del formulario se realiza con la tecla de tabulación, pero el usuario está más habituado al uso de la tecla Intro, por lo que a continuación se va a presentar una técnica que permite navegar por los distintos campos del formulario utilizando la tecla Intro.

Se utiliza el evento keypress en cada campo desde el que debe avanzar el foco a otro campo del formulario. El manejador de eventos llamará a la función siguienteFoco(), que se presenta a continuación:

```
<input type="text" name="campo1" id="campo1" onkeypress="return siguienteFoco(this.form,
 'campo2', event)">
```

La función siguienteFoco() realiza lo siguiente:

- En primer lugar, se debe comprobar que la tecla presionada es Intro, que tiene los códigos ASCII 13 y 3.
- En segundo lugar, aplica el foco al campo siguiente que ha sido pasado como parámetro de la función:

```
function siguienteFoco(form, siguienteCampo, evt) {
   evt = (evt) ? evt : event;
   var codCaracter = (evt.charCode) ? evt.charCode :
      ((evt.which) ? evt.which : evt.keyCode);
   if (codCaracter == 13 || codCaracter == 3) {
      form.elements[siguienteCampo].focus();
      return false;
   }
   return true;
}
```

Una de las posibles potencialidades del uso de funciones como la anterior es la posibilidad de poder adaptar la introducción de los datos en un orden distinto al que se sigue en la distribución de los campos en la página HTML.

2.1.3. Enviar formularios con la tecla Intro

En este subapartado, se va a presentar un conjunto de técnicas que permiten, junto con lo presentado en el subapartado anterior, simular un formulario de aplicación, es decir, un formulario en el que el desplazamiento por los distintos campos se realiza con la tecla Intro, y cuando en el último campo de texto se pulsa Intro, se produce el evento de envío del formulario.

Por lo tanto, en el último campo del formulario se deberá llamar a una función que acabe con una llamada al método submit() del objeto Form. A continuación, se presenta un ejemplo de esta función:

```
<html>
<head>
</head>
<body>
<form id="formulari">
    <P>
    <LABEL for="nombre">Nombre: </LABEL>
              <INPUT type="text" id="nombre"><BR>
    <LABEL for="apellido">Apellido: </LABEL>
              <INPUT type="text" id="apellido"><BR>
    <LABEL for="email">email: </LABEL>
              <INPUT type="text" id="email"><BR>
    <INPUT type="radio" name="sexo" value="Varón" onkeypress="return tramesaAmbEnter(event)">
     Varón<BR>
    <INPUT type="radio" name="sexo" value="Mujer" onkeypress="return tramesaAmbEnter(event)">
     Mujer<BR>
    </P>
 </form>
<script type="text/javascript">
function envioConEnter(evt){
       evt = (evt) ? evt : event;
       var destino = (evt.target) ? evt.target : evt.srcElement;
       var codCaracter = (evt.charCode) ? evt.charCode : ((evt.which) ? evt.which : evt.keyCode);
       if (codCaracter == 13 || codCaracter == 3) {
        formulari.submit();
 }
}
</script>
</body>
</html>
```

De esta manera, en el último campo del formulario se realizará la siguiente asignación:

```
onkeypress="return envioConEnter(event)"
```

Un detalle que considerar es que el objeto Form debe tener definido el manejador de eventos onsubmit="return false", ya que de esta manera se obliga a que sólo el método submit() del script sea capaz de realizar el envío.

2.1.4. Tabulación automática

Existen formularios en los que cierta información se introduce en un conjunto combinado de campos de texto, que se caracterizan por una longitud definida en cada uno de ellos.

Campos de texto con longitud definida

Un ejemplo muy común son los cuadros de texto en los que se introducen los códigos de una tarjeta de crédito o de una cuenta bancaria.

En este tipo de formularios, cuando el usuario finaliza la introducción de los números en uno de los campos, automáticamente el cursor se sitúa en el siguiente campo de texto. El ejemplo que se presenta a continuación muestra una de las maneras en las que se puede implementar el efecto anterior y lo va a realizar en un formulario con 4 campos de tamaño máximo 4:

```
<html>
<head>
</head>
<body>
<form id="formulario" onsubmit="return false">
Número de la tarjeta:
<input type="text" id="nt1" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt2', event)" />
<input type="text" id="nt2" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt3', event)" />
<input type="text" id="nt3" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt4', event)" />
<input type="text" id="nt4" size="5" maxlength="4" />
 </form>
<script type="text/javascript">

function mueveFocus(campo, siguiente, evt){
       evt = (evt) ? evt : event;
       var codCaracter = (evt.charCode) ? evt.charCode : ((evt.keyCode) ? evt.keyCode :
       (( evt.which) ? evt.which : 0));
       if (codCaracter >31 && camp.value.length == camp.maxLength) {
campo.forma.elementos[siguiente].focus();
       }
}

</script>
</body>
</html>
```

El manejador de la función keyup llama a la función mueveFoco() que se encarga de pasar el foco al siguiente campo.

Se puede observar que el último campo no llama a la función mueveFoco(). Se podría implementar para que saltara el foco al siguiente campo del formulario, simplemente pasándole el valor en el segundo parámetro de la función.

2.2. Control de los campos del formulario

Existen dos mecanismos para evitar o bloquear el acceso a ciertos campos de un formulario:

a) Desactivar un campo, de manera que éste aparece visible al usuario, pero no está activo, lo que implica que no se pueda introducir texto ni que éste se pueda modificar.

b) Ocultar un campo. En este caso, el campo desaparece del formulario y no es visible para el usuario final, aunque el campo sí que existe realmente.

En este subapartado, se van a estudiar los dos mecanismos que permiten desactivar y ocultar campos mediante unos ejemplos sencillos.

2.2.1. Desactivar campos

La desactivación de campos de formulario se implementa a partir de la propiedad disabled, que admite un valor booleano:

```
document.formulario.campoTexto.disabled = true;
```

El código anterior desactivaría el campo "campoTexto", de manera que no sería accesible para el usuario y, en general, mostraría un aspecto sombreado de color gris. Aunque los scripts pueden leer y escribir valores en los campos desactivados, éstos no son enviados al servidor si no se activan.

La activación del anterior campo se realizaría con la siguiente sentencia:

```
document.formulario.campoTexto.disabled = false;
```

2.2.2. Ocultar y mostrar controles

Es posible ocultar ciertos campos de un formulario, de manera que sólo se muestren en el caso de que se cumpla cierta condición, por ejemplo que el usuario haya seleccionado alguna opción en algún campo de selección o marque una casilla de selección.

A continuación, se presenta un ejemplo de formulario que dispone de un conjunto de campos ocultos, que son mostrados al usuario final cuando éste responde afirmativamente a la tercera pregunta:

```
<form name="cuestionario">
```

```
...
<p>3. ¿Quieres conocer los nuevos modelos?<br />
<input type="radio" id="resp0" name="respuesta" onclick="gestDecision(event)"
/>No
<input type="radio" id="resp1" name="respuesta" onclick="gestDecision(event)"
/>Si
<div id="conocerModelos" style="display:none; margin-left:20px">
</p>
<p>
3a. ¿Cuál es tu marca preferida?
<select name="modelos">
   <option value="">Elige uno:</option>
   <option value="1">Ferrari</option>
   <option value="2">Mercedes</option>
   <option value="3">BMW</option>
</select>
</p>
</div>
<p>4. ¿Cuántos años tiene tu actual vehículo?
<select name="edadVehiculo">
   <option value="">Elige una opción:</option>
   <option value="1">Menos de 3 años:</option>
   <option value="2">Entre 3 y 7 años</option>
   <option value="3">Más de 7 años</option>
</select>
</p>
...
</form>
```

En el código anterior se ha utilizado una capa para introducir en ella la lista desplegable de las marcas. De esta manera, en principio, la lista está oculta al establecer el atributo display a none de la capa, no se muestra inicialmente y es el manejador del evento click de las opciones radio el que cambiará el estado de la capa:

```
function gestDecision(evt) {
   evt = (evt) ? evt : event;
   var campo = (evt.target) ? evt.target : evt.srcElement;
   var capa = document.getElementById("conocerModelos");
   if (campo.id =="resp1") {
      capa.style.display = "block";
   } else {
      capa.style.display = "none";
   }
}
```

Como se observa en la función, se utiliza el objeto Event para capturar el campo y, a continuación, se asigna a la variable *capa* el contenedor de los campos, de manera que si el campo seleccionado es "resp1", se muestra la capa oculta o siguen los campos ocultos en caso contrario.

2.2.3. Permitir sólo un tipo de datos en un campo

En ciertas aplicaciones web, es interesante restringir el tipo de datos que se puede introducir en un campo de texto. Posteriormente, se plantean ejemplos que definen una función, que es llamada desde el evento keypress.

En primer lugar, la siguiente función restringe la entrada de valores a los números entre el 0 y el 9, además, permite la introducción de valores ASCII menores que el 32, ya que entre estos valores se encuentran los caracteres alfanuméricos que incluyen las teclas Retroceso 8, Tabulador 9 e Intro 13:

```
function soloNumeros(evt){
   evt = (evt) ? evt : event;
   var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.wich) ?
    evt.which : 0;
   if (codCaracter > 31 && (codCaracter < 48 || codCaracter > 57)) {
      alert("Introduce sólo valores numéricos!!!");
      return false;
   }
   return true;
}
```

En el siguiente ejemplo se plantea una función que valida la introducción de varias letras. Para ello, se debe tener en cuenta el hecho de que las mayúsculas y minúsculas tienen distintos códigos y además no son correlativos. La siguiente función realiza la comprobación de la introducción de letras, tanto en mayúsculas como en minúsculas:

```
function soloLetras(evt){
   evt = (evt) ? evt : event;
   var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ?
     evt.which : 0;
   if (codCaracter > 31 && (codCaracter < 65 || codCaracter > 90) && (codCaracter <
    97 || codCaracter > 122)) {
      alert("Introduce sólo letras!!!");
      return false;
   }
   return true;
}
```

La misma estructura de función sirve para definir una función que obligue a la entrada de ciertos valores simples, como "Y" o "N", "0" o "1", o cualquier combinación que se defina. Por ejemplo, la siguiente función sólo permite la introducción de los valores "Y" o "N", tanto en mayúsculas como en minúsculas:

```
function soloYoN(evt){
   evt = (evt) ? evt : event;
   var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ? evt.which : 0;
   if (codCaracter > 31 && codCaracter != 78 && codCaracter != 89 &&
   codCaracter != 110 && codCaracter != 121) {
      alert("Introduce sólo los valores 'Y' o 'N'");
      return false;
   }
   return true;
}
```

En un ejemplo como el anterior, se puede reforzar la seguridad limitando el tamaño del campo a un único carácter:

```
<input type="text" name="valor" size="2" maxlength="1" onkeypress="return
 soloYoN(event)" /> (Y/N)
```

A continuación, se presenta la tabla de códigos ASCII:

Tabla 1. Códigos ASCII

Caracteres no imprimibles				**Caracteres imprimibles**								
Nombre	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**
Nulo	0	00	NUL	32	20	Espacio	64	40	@	96	60	`
Inicio de cabecera	1	01	SOH	33	21	!	65	41	A	97	61	a
Inicio de texto	2	02	STX	34	22	"	66	42	B	98	62	b
Fin de texto	3	03	ETX	35	23	#	67	43	C	99	63	c
Fin de transmisión	4	04	EOT	36	24	$	68	44	D	100	64	d
enquiry	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	6	06	ACK	38	26	&	70	46	F	102	66	f
Campanilla (beep)	7	07	BEL	39	27	'	71	47	G	103	67	g
backspace	8	08	BS	40	28	(	72	48	H	104	68	h

Caracteres no imprimibles				Caracteres imprimibles								
Nombre	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**	**Dec**	**Hex**	**Car.**
Tabulador horizontal	9	09	HT	41	29	)	73	49	I	105	69	i
Salto de línea	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
Tabulador vertical	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
Salto de página	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
Retorno de carro	13	0D	CR	45	2D	-	77	4D	M	109	6D	m
Shift fuera	14	0E	SO	46	2E	.	78	4E	N	110	6E	n
Shift dentro	15	0F	SI	47	2F	/	79	4F	O	111	6F	o
Escape línea de datos	16	10	DLE	48	30	0	80	50	P	112	70	p
Control dispositivo 1	17	11	DC1	49	31	1	81	51	Q	113	71	q
Control dispositivo 2	18	12	DC2	50	32	2	82	52	R	114	72	r
Control dispositivo 3	19	13	DC3	51	33	3	83	53	S	115	73	s
Control dispositivo 4	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	21	15	NAK	53	35	5	85	55	U	117	75	u
Sincronismo	22	16	SYN	54	36	6	86	56	V	118	76	v
Fin bloque transmitido	23	17	ETB	55	37	7	87	57	W	119	77	w
Cancelar	24	18	CAN	56	38	8	88	58	X	120	78	x
Fin medio	25	19	EM	57	39	9	89	59	Y	121	79	y
Sustituto	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
Escape	27	1B	ESC	59	3B	;	91	5B	[	123	7B	{

Caracteres no imprimibles				Caracteres imprimibles								
Nombre	Dec	Hex	Car.	Dec	Hex	Car.	Dec	Hex	Car.	Dec	Hex	Car.
Separador archivos	28	1C	FS	60	3C	<	92	5C	\	124	7C	|
Separador grupos	29	1D	GS	61	3D	=	93	5D	]	125	7D	}
Separador registros	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
Separador unidades	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

2.3. Validación de campos

En este último subapartado, se va a mostrar una serie de funciones que realizan un conjunto de validaciones sobre campos de formulario. El lanzamiento de estas funciones se ejecuta con el manejador de evento change. Por ejemplo:

```
<input type = "text" size = "30" id = "correoE" onchange =
 "esCorreoOK(this)" />
```

A continuación, se van a presentar cuatro funciones:

- noVacio(): comprueba que el campo de texto contiene al menos un carácter;
- esNumerico(): comprueba que el contenido del campo de texto es un valor numérico;
- longitud(): comprueba que el campo contiene exactamente 12 caracteres;
- esCorreoE(): comprueba que el contenido del campo tiene el formato de una dirección de correo electrónico.

La función noVacio() comprueba que el campo tiene al menos un carácter:

```
function noVacio(elem) {
   var cadena = elem.value;
   if(cadena == null || cadena.length == 0) {
      alert("Atención, el campo es obligatorio.")
      return false;
   } else {
      return true;
   }
```

```
}
```

La función anterior realiza dos validaciones sobre el contenido del campo: que contenga un valor nulo o una cadena de texto vacía, ya que se trata de dos opciones similares pero ambas se deben comprobar.

El siguiente ejemplo comprueba que el valor introducido es numérico:

```
function esNumerico(elem) {
   var cadena = elem.value;
   var unDecimal = false;
   var unCaracter = 0;
   //Se asegura que el contenido está en formato string
   cadena = cadena.toString();
   for (var i = 0; i < cadena.length; i++) {
      unCaracter = cadena.charAt(i).charCodeAt(0);
      // Se comprueba si tiene un signo negativo en el primer carácter
      if (unCaracter == 45) {
      if (i == 0) {
         continue;
         } else {
            alert("Sólo el primer caracter puede ser el signo negativo.");
            return false;
         }
      }
      // Se comprueba el punto decimal
      if (unCaracter == 46) {
         if (!unDecimal) {
            unDecimal = true;
            continue;
         } else {
            alert("Sólo un.decimal en un número.");
            return false;
         }
      }
      //No se admiten caracteres fuera del rango 0-9
      if (unCaracter < 48 || unCaracter > 57) {
         alert("Introduce sólo números.");
         return false;
      }
   }
   return true;
}
```

El refinamiento de la función anterior se basa en la validación, si el primer carácter es el signo negativo (que marca que el número es negativo) y existe un punto intercalado que indica que el número tiene valores decimales.

El ejemplo que se presenta a continuación comprueba la longitud del contenido de un campo de texto:

```
function longitud(elem,long) {
   var cadena = elem.value;
   var long = parseInt(long);
   if (cadena.length != long) {
      alert("El campo no contiene los "+ long +" caracteres requeridos");
      return false;
   } else {
      return true;
   }
}
```

El ejemplo anterior es muy simple, se basa en la propiedad length del objeto String, pero puede ser muy útil en ciertos formularios.

Para finalizar el apartado se mostrará un ejemplo de validación de un campo que debe contener una dirección de correo electrónico:

```
function esCorreoE(elem) {
   var cadena = elem.value;
   cadena = cadena.toLowerCase();
   if (cadena.indexOf("@") > 1) {
      var addr = cadena.substring(0, cadena.indexOf("@"));
      var dominio = cadena.substring(cadena.indexOf("@") + 1, cadena.length);
      // Al menos es necesario un dominio en la cadena de la dirección
      if (dominio.indexOf(".") == -1) {
         alert("Verifica que el dominio sea correcto");
         return false;
      }
      // Se revisa carácter por carácter
      for (var i = 0; i < addr.length; i++) {
         unCaracter = addr.charAt(i).charCodeAt(0);
         // El punto no esta permitido en esta porción
         if ((i == 0 && (unCaracter == 45 || unCaracter == 46)) ||
         (i == addr.length - 1 && unCaracter == 46)) {
            alert("Verifica la porción del nombre de usuario");
            return false;
         }
         //Se comprueban caracteres validos (- . _ 0-9 a-z)
         if (unCaracter == 45 || unCaracter == 46 || unCaracter == 95 ||
            (unCaracter > 47 && unCaracter < 58) ||
```

```
            (unCaracter > 96 && unCaracter < 123)) {
            continue;
         } else {
            alert("Verifica la porción del nombre de usuario.");
            return false;
         }
      }
      for (i = 0; i < dominio.length; i++) {
         unCaracter = dominio.charAt(i).charCodeAt(0);
         if ((i == 0 && (unCaracter == 45 || unCaracter == 46)) ||
         ((i == dominio.length - 1 || i == dominio.length - 2) &&
         unCaracter == 46)) {
            alert("Verifica el dominio de la dirección.");
            return false;
         }
         if (unCaracter == 45 || unCaracter == 46 || unCaracter == 95 ||
         (unCaracter > 47 && unCaracter < 58) ||
         (unCaracter > 96 && unCaracter < 123)) {
            continue;
         } else {
            alert("Verifica el dominio de la dirección.");
            return false;
         }
      }
      return true;
   }
   alert("La dirección de correo no está correctamente formateada. Por favor, compruébala.");
   return false;
}
```

La función anterior se basa en las siguientes comprobaciones:

- En primer lugar, se comprueba que la "@" se encuentra en la cadena de texto; en caso contrario, no se sigue con la validación.

- En segundo lugar, se comprueba que el "." que define el dominio de primer nivel se encuentra en la cadena de texto; en caso contrario, no se sigue.

- En tercer lugar, se recorre carácter por carácter la primera parte de la cadena de texto que corresponde al usuario de la cuenta de correo, y se comprueba que está formada por caracteres válidos en una cuenta de correo.

- En cuarto lugar, se recorre la parte correspondiente del dominio de la cuenta de correo y se valida que dispone de un dominio y de un subdominio.

Se trata de una validación de correo electrónico muy sencilla. Se puede añadir mayor complejidad introduciendo más comprobaciones, como la validación de que los dominios fueran reales a partir de una lista. Esta función puede ser llamada a partir del evento onblur del campo que ha de contener la dirección electrónica.

3. Posicionamiento dinámico

La distribución de los elementos en una página web se rige por el orden en el que se escriben en el código HTML. Hasta la aparición de las capas, la única manera de posicionar elementos era usando tablas. En cualquier caso, una vez que se ha cargado la página, los elementos que la forman son estáticos.

La especificación CSS introduce ciertos atributos que permiten posicionar objetos en la página (también conocido como CSS-P o DHTML). Con el uso de DHTML o CSS-P, se rompe la limitación anterior, ya que:

- Una vez cargada la página, es posible modificar los atributos de posicionamiento utilizando eventos o scripts definidos por el programador.
- Es posible solapar elementos HTML en la página web.
- Es posible ocultar y mostrar elementos HTML en la página web.
- Es posible mover elementos e incluso realizar ciertas animaciones.

Estas características se introdujeron a partir de las versiones 4.x de los navegadores, lo que generó problemas de interpretación entre los dos principales navegadores de aquella época. De hecho, se trataba del área en la que la divergencia era mayor.

3.1. Propiedades CSS de posicionamiento

Las propiedades de posicionamiento permiten que un elemento HTML se posicione en cualquier coordenada del documento. Se suele denominar **capa** a las áreas de posicionamiento que suelen crearse con la etiqueta HTML `<div>` y cuyas propiedades o atributos se presentan a continuación:

1) position

Esta propiedad indica el tipo de posicionamiento utilizado en el elemento. Los valores disponibles para esta propiedad son los siguientes:

- static: es el valor predeterminado e indica que el elemento no es posicionable.
- absolute: este valor provoca que el elemento queda fuera del flujo de HTML y su posición queda relativa a la esquina superior izquierda de la página.

Los objetos pueden quedar superpuestos sobre otros elementos del flujo de la página.

- relative: este valor provoca que el elemento ocupe el mismo lugar que ocuparía si no estuviera posicionado y se desplaza de su posición el valor que se indique.

2) top y left

Estos dos valores definen la posición de la capa, de manera que se define desde la parte superior izquierda de la región que lo incluye (normalmente la región coincide con la ventana del navegador).

3) height y width

Definen el tamaño de la capa a partir de la altura y anchura, que debe ser medida en píxeles o en porcentajes.

4) clip

La propiedad clip permite definir un área rectangular dentro de un elemento. Todo lo que quede dentro de este clip será visible, y lo de fuera, invisible.

La región se define indicando las propiedades top, right, bottom y left del rectángulo, que deben ser indicadas en píxeles y tomar como referencia el elemento contenedor y no la página completa. La sintaxis es la siguiente:

```
clip: rect(top right bottom left);
```

Veamos un ejemplo de su uso:

```
<div id="cortada" style="position:absolute; left:100; top:250; width: 400; height:
100; background-color: #CC9966; layer-background-color: #CC9966; border:
1px none #000000; clip: rect(30 330 80 130)" >
```

Se debe tener en cuenta que el espacio reservado para toda la capa se mantiene inalterado, aunque no sea visible. Además, es posible mover el clip modificando los parámetros de posicionamiento de la capa sobre la que está definido para conseguir el efecto deseado.

5) visibility

El atributo visibility también tiene un significado bastante obvio: un elemento con la regla visibility: hidden no será visible.

Los valores que puede tomar la propiedad visibility son:

- hidden: oculta la capa.
- visible: muestra la capa, si está visible.
- inherit: indica si la visibilidad se hereda del contenedor superior.

La utilidad de tener un elemento invisible no quedaría clara si no fuera por el hecho de que la visibilidad del elemento se puede modificar por eventos o por un script definido por el usuario. Gran parte de los menús desplegables realizados en JavaScript se basan en esta propiedad.

6) z-index

Define el orden de solapamiento de las áreas ocupadas por los elementos, de manera que una capa con valor z-index mayor se sitúa sobre una capa con un valor z-index menor. Si no se asigna el valor z-index, el orden de solapamiento lo define el mismo orden de creación de las capas, y el último elemento definido es el que queda en primer plano.

3.2. Un poco de historia: Navegadores 4.x

Con la aparición de CSS-P en los navegadores 4.x, la mayor parte de los problemas fueron debidos a que Netscape 4 introdujo una etiqueta propietaria `<layer>` que no era compatible con IE 4 y que fue abandonada en las posteriores versiones del navegador.

3.2.1. Capas en Netscape 4

Como se ha comentado, Netscape introdujo una nueva etiqueta propietaria `<layer>` que proporcionaba mecanismos de posicionamiento similares a los del estándar CSS. La siguiente tabla muestra las propiedades del objeto layer relacionadas con el posicionamiento dinámico.

Tabla 2. Propiedades del objeto layer

Propiedad	Descripción
clip	Referencia al objeto clip de recorte de la capa. Este objeto se define con las propiedades top, right, bottom y left.
left	Posición de la coordenada x de la capa.
top	Posición de la coordenada y de la capa.
pageX	Posición de la coordenada x de la capa relativa a la página.
pageY	Posición de la coordenada x de la capa relativa a la página.
visibility	Indica la visibilidad de la capa, admite los valores show y hide.
zIndex	Indica el orden de solapamiento de la capa.

A continuación, se presenta un ejemplo de capa creada para Netscape 4.x:

```
<layer name="capaNN" pageX="100" pageY="100" width="100" height="50"
bgcolor="#ffff99">Capa creada con layer! </layer>
```

La etiqueta <layer> nació y murió en las versiones 4 del Netscape; como se ha comentado, no fue soportada a partir de versiones posteriores del navegador ni por ningún navegador de la competencia.

Por otra parte, justo antes de la presentación del Netscape 4, el navegador adoptó soporte para las etiquetas posicionadas, por lo tanto, la capa anterior también se puede crear con la siguiente sintaxis:

```
<div id="capaNN" style="position: absolute; top: 100px; left: 100px; width: 100px;
height: 50px; background-color: #ffff99">Capa creada con div!</div>
```

De hecho, el acceso a las capas creadas con la etiqueta se realizaba utilizando la colección layers[ ]. Por ejemplo, el acceso a la capa anterior se realizaba con la siguiente sintaxis:

```
document.layers['capaNN'];
```

Pero, sin lugar a dudas, la característica que hizo perder terreno a Netscape fue que éste no reflejaba de manera dinámica los cambios producidos en la propiedad style, lo que implica que no se podían modificar las propiedades de manera dinámica. Es decir, si se modificaba un atributo de la propiedad style de la etiqueta o cualquier propiedad de la etiqueta <layer> mediante un script, esta modificación no se reflejaba en la página.

De esta manera, el posicionamiento dinámico en Netscape 4.x no se basaba en la modificación de las propiedades "top" y "left", sino a partir de un conjunto de métodos implementados en el objeto layer. A continuación, se pueden ver los principales métodos:

- moveBy(incrementoX,incrementoY): provocaba un desplazamiento de la capa a partir de los valores pasados como parámetros;
- moveTo(x,y): situaba la capa en las coordenadas indicadas en los parámetros x e y;
- moveToAbsolute(x,y): situaba la capa en las coordenadas absolutas indicadas en los parámetros x e y;
- moveAbove(capa): situaba la capa actual sobre la capa pasada como parámetro;
- moveBelow(capa): situaba la capa actual bajo la capa pasada como parámetro;

- resizeBy(incrementoX,incrementoY): incrementaba el tamaño de la capa con los valores pasados en los parámetros incrementoX e incrementoY;

- resizeTo(alto,ancho): especifica el tamaño de la capa con los valores pasados en los parámetros "alto" y "ancho".

3.2.2. Capas en Internet Explorer 4

Las capas en Internet Explorer se definen utilizando la etiqueta `<div>` a la que se define la propiedad style con cada uno de sus atributos. Veamos a continuación un ejemplo:

```
<div id="capaIE" style="position: absolute; top: 100px; left: 100px; width:
100px; height: 100px; background-color: "#ffff99">Capa creada con div!</div>
```

La principal diferencia del navegador Internet Explorer frente a Netscape se basaba en que el primero sí que actualizaba la presentación ante cualquier cambio de la propiedad style de manera dinámica. Así pues, desde un script se podía modificar cualquier atributo de la etiqueta `style` y éste se representaba en la página de manera inmediata.

Por lo tanto, en Internet Explorer las modificaciones de las capas no se realizaban a partir de métodos del objeto, sino a partir de la modificación de las propiedades del objeto style. Por ejemplo, a partir de la capa anterior, el siguiente código:

```
document.all['capaIE'].style.left += 10;
document.all['capaIE'].style.backgroundColor = 'orange';
```

provoca un desplazamiento de la capa a la derecha de 10 píxeles y modifica el color de fondo de ésta.

Una de las características que hay que tener en cuenta cuando se usen los atributos CSS en JavaScript es que la sintaxis de los atributos cambia cuando éstos se utilizan desde JavaScript. Es decir, en el ejemplo anterior en la creación de la etiqueta `<div>` se ha especificado el color de fondo mediante el atributo background-color, mientras que en el código script esta propiedad del objeto style se denomina backgroundColor.

La regla general consiste en convertir los atributos compuestos de CSS en una única palabra en el DOM, utilizando lo que se conoce como joroba de camello (la segunda palabra cambia su primera letra a mayúsculas y elimina el guión que los separa).

```
background-color - backgroundColor
```

3.3. CSS-P en navegadores DOM W3C

El acceso a las capas en los navegadores que cumplen la especificación DOM de W3C se realiza con métodos de acceso ya explicados. De esta manera, cuando el nodo (o capa) ya está referenciado en una variable, la manipulación de sus atributos CSS se realiza a partir de su objeto style (de manera similar a la que se implementó en Internet Explorer).

Ved también

Los métodos de acceso a las capas en los navegadores que cumplen la especificación DOM de W3C se trataron en el apartado 2 del módulo "Introducción al DOM" de esta asignatura.

Por ejemplo, dada la siguiente capa:

```
<div id="capaDOM" style="position: absolute; top: 100px; left: 100px; width: 100px;
 height: 100px; background-color: "#ffff99">Capa creada con div!</div>
```

La modificación de su color de fondo se realiza con la siguiente sintaxis:

```
document.getElementById("capaDOM").style.backgroundColor = "orange";
```

De esta manera, es posible manipular las propiedades del objeto style de cualquier objeto del documento, pero la manipulación de reglas definidas mediante selectores en una hoja de estilo externa se realiza a partir de una colección styleSheets[ ] definida en un objeto Document y que se especifica en el DOM2 para CSS. Mediante esta colección, se accede a las reglas de bloque definidas y se manipulan directamente mediante los métodos insertRule() y deleteRule().

3.4. Posicionamiento cross-platform

Es necesario que, mediante funciones de script o mediante eventos provocados por el usuario, las propiedades de posicionamiento cambien. Se quiere modificar la posición, el tamaño, la visibilidad o el índice z de los objetos. Además, es necesario que lo que se haga sea funcional en todos los navegadores o al menos en los más utilizados.

Como se ha visto en los subapartados anteriores, los mecanismos con los que se accede y se interactúa con los objetos en Internet Explorer 4 y en Netscape 4 y en los navegadores basados en el DOM son distintos. Afortunadamente, este problema tiene solución y además la solución es abierta, en el sentido de que, si aparecen nuevos problemas con versiones posteriores de los navegadores, se pueden tratar siguiendo la misma estrategia.

La programación cross-platform se basa en la programación condicionada al navegador que lee el script, de esta manera, una vez detectado el navegador y dependiendo de las características de éste, se ejecuta un código u otro.

3.4.1. Detección del navegador

En el caso de que las funciones que se definen dependan del navegador que las ejecute, se puede utilizar la técnica explicada en el apartado 1.1.1 para detectar el navegador y utilizar una estructura condicional a las funciones.

A continuación se va a presentar una librería con las siguientes funciones:

```
function hide(nombreCapa) {}
function show(nombreCapa){}
function setX(nombreCapa,coorX){}
function setY(nombreCapa,coorY){}
function setZ(nombreCapa, indiceZ){}
function setHeight(nombreCapa, altura){}
function setWidth(nombreCapa, ancho){}
```

Para ello, en primer lugar se crea una función que es utilizada por las funciones anteriores y cuyo objetivo es la asignación a una variable la referencia de la capa, ya que cada navegador tiene un mecanismo distinto para realizar una referencia a las capas:

```
function getElement(nombreCapa){
   return document.getElementById(nombreCapa);
}
```

Como se observa, en la función la recuperación de la capa es inmediata.

Definida la función anterior, las funciones de la API son muy sencillas:

```
function hide(nombreCapa){
   //Se recupera la referencia a la capa y se asigna el atributo hidden
   var capa = getElement(nombreCapa);
   capa.style.visibility = 'hidden';
}

function show(nombreCapa){
   //Se recupera la referencia a la capa y se asigna el atributo visible
   var capa = getElement(nombreCapa);
   capa.style.visibility = 'visible';
}

function setX(nombreCapa,x){
   //Se recupera la referencia a la capa y se asigna la coordenada
   var capa = getElement(nombreCapa);
   capa.style.left=x+"px";
}
```

```
function setY(nombreCapa, y){
   //Se recupera la referencia a la capa y se asigna la coordenada
   var capa = getElement(nombreCapa);
   capa.style.top=x+"px";
}

function setZ(nombreCapa, zIndex){
   //Se recupera la referencia a la capa y se asigna el valor
   var capa = getElement(nombreCapa);
   capa.style.zIndex = zIndex;
}

function setHeight(nombreCapa, height){
   //Se recupera la referencia a la capa y se asigna el valor
   var capa = getElement(nombreCapa);
   capa.style.height = height;
}

function setWidth(nombreCapa, width){
   //Recuperamos la referencia a la capa y se asigna el valor
   var capa = getElement(nombreCapa);
   capa.style.width = width;
}
```

De esta manera, las funciones anteriores se escriben en un fichero externo que es referenciado en el script cuando son necesarias y pueden ampliarse con más funciones o pueden introducirse estructuras condicionales en su interior si la asignación de los valores no se realiza con la misma sintaxis en algún navegador. Es decir, si nuevas versiones aportan propiedades no compatibles con las anteriores, se pueden modificar las funciones anteriores para que soporten las nuevas características de estos navegadores.

3.5. Fundamentos de la animación

La animación en JavaScript es muy sencilla y se basa en un pequeño conjunto de técnicas basadas en propiedades CSS de las capas y en el manejo del tiempo con los métodos existentes en el lenguaje. De esta manera, los 4 factores que se deben tener en cuenta son:

- Posicionamiento.
- Definición del recinto.
- Definición del movimiento.
- Temporalización.

En los siguientes subapartados, se muestra cómo se pueden implementar las características anteriores.

3.5.1. Posicionamiento

El posicionamiento es la acción de situar o mover la capa por un cierto escenario definido; en el caso de que la asignación de posiciones absolutas fueran suficiente las funciones setX() y setY() definidas en el subapartado anterior, ya resolverían esta primera parte del problema.

El problema radica en que el movimiento se basa en el incremento o decremento de la posición actual de la capa; por lo tanto, es necesario obtener la posición actual de la capa y a partir de estos valores, incrementar o disminuir con las unidades definidas para ser a posteriori asignados mediante las funciones setX() y setY().

Por lo tanto, es necesario disponer de dos funciones getX(nombreCapa) y getY(nombreCapa) que proporcionen la posición actual de la capa. El código de estas funciones es muy simple, como se puede observar a continuación:

```
function getX(nombreCapa){
   var capa = getElement(nombreCapa);
   return(parseInt(capa.style.left))
}

function getY(nombreCapa){
   var capa = getElement(nombreCapa);
   return(parseInt(capa.style.top))
}
```

De esta manera, con las anteriores funciones se pueden desplazar 10 píxeles a la derecha una capa de nombre "bola" con el siguiente código:

```
posX = getX("bola");
setX("bola",posX+10);
```

Pero el desplazamiento debe tener unos límites en cuanto a la longitud de éste, así como en cuanto al recinto en el que se puede mover. Estos dos aspectos se estudian a continuación.

3.5.2. Paso y tamaño del recinto

En el ejemplo del subapartado anterior se ha desplazado la capa 10 píxeles hacia la derecha: este valor se conoce como **paso de la capa**. En una animación se definirá una variable con el valor de paso, de manera que el valor del paso podrá ser modificado dinámicamente a partir de un evento de usuario o cuando se cumpla cierta condición, como la llegada al extremo del recinto.

Se debe tener en cuenta que el paso indica el sentido del desplazamiento, de manera que en ciertas situaciones el sentido del desplazamiento se debe invertir, y esto es tan sencillo como el cambio de signo del paso. De esta manera, si paso = 10 el siguiente código:

```
paso = paso*-1;
```

actualizará paso al valor –10 invirtiendo con este simple código el sentido del movimiento, tanto en horizontal como vertical.

Ahora bien, si la animación necesita que los movimientos en los sentidos vertical y horizontal sean independientes, se pueden definir dos variables *pasoX* y *pasoY*.

Otro aspecto que se debe considerar son los límites del recinto en los que se va a realizar la animación, de manera que si se está programando una animación autónoma (sin intervención del usuario), cuando la capa llegue al límite del recinto ésta debe o bien situarse en otro extremo del recinto o bien cambiar el sentido y/o dirección del movimiento (variable *paso*).

La estrategia de definición del recinto se basa en cuatro nuevas variables que definen los límites, por ejemplo:

```
var limiteSup = 100;
var limiteInf = 400;
var limiteIzq = 100;
var limiteDer = 600;
```

La función de estas variables es comprobar en cada modificación de la posición de la capa (movimiento) si ésta ha llegado a uno de estos límites, y en caso afirmativo, realizar las acciones definidas para que no supere los límites (cambiar el sentido, saltar a otra posición, parar el movimiento, etc.).

Es posible que en ciertas ocasiones estos límites dependan de las dimensiones del área del documento o de la pantalla, por lo que se deberán utilizar ciertas propiedades de los objetos Document o Screen en la definición de los límites del recinto.

3.5.3. Temporalización

La temporalización es especialmente importante en las animaciones autónomas, ya que el movimiento se basa en la llamada cada equis tiempo a una función que provoca un cambio de posición de un objeto. Para este objetivo, se suele utilizar la función setInterval(nombreFuncion, miliSegundos).

La función anterior dispone de dos parámetros: el primero contiene el nombre de la función que se debe ejecutar y el segundo es el número de milisegundos que pasarán entre dos llamadas a la función. Por ejemplo:

```
var repite = setInterval("mueve()",1000);
```

Ejecuta la función mueve() al cabo de un segundo.

Por otra parte, en cierto momento será necesario que la función mueve() deje de ejecutarse, y para ello se debe cancelar la función setInterval().

La cancelación de esta función se realiza mediante la función clearInterval(variableInterval), que debe recibir como parámetro la variable que contiene la referencia a la función setInterval().

Por ejemplo, la cancelación de la función definida en el ejemplo anterior se realizaría de la siguiente manera:

```
clearInterval(repite);
```

3.6. Aplicaciones simples de CSS-P

En este subapartado se van a estudiar cuatro sencillos ejemplos que utilizan como base técnica lo explicado en los subapartados anteriores.

3.6.1. Animación en línea recta

Se va a definir una animación en la que se desplazará un elemento desde una posición determinada de una página a otra posición, siguiendo una línea recta entre ambas. Para ello, se va a definir una función animaRecta() que necesitará los siguientes parámetros de entrada:

- El identificador de la capa que se va a mover.
- La coordenada x de la posición inicial.
- La coordenada y de la posición inicial.
- La coordenada x de la posición final.
- La coordenada y de la posición final.
- La velocidad del movimiento.

De esta manera, la siguiente llamada a la función:

```
animarecta("capa1", 100, 100, 300, 300, 10);
```

moverá la capa "capa1" desde la posicioón inicial (100, 100) hasta la posición final (300, 300) siguiendo una línea recta y a 10 de velocidad.

```
// Se crea un objeto anima que dispone de un conjunto
```

```
//de propiedades relacionadas con el movimiento
var anima = new Object();
//Se inicializa el objeto anima
function initAnima() {
   anima = {capa:"", valorX:0, valorY:0, finX:0, finY:0, pasoX:0, pasoY:0, distX:0,
   distY:0, MoviX:0, MoviY:0, vel:1, cami:1, interval:null };
}
// Se define la función animaRecta
function animaRecta(capa, iniciX, iniciY, finalX, finalY, pas) {
   initAnima();
   anima.capa = capa;
   anima.valorX = iniciX;
   anima.valorY = iniciY;
   anima.finX = finalX;
   anima.finY = finalY;
   anima.distX = Math.abs(finalX - iniciX);
   anima.distY = Math.abs(finalY - iniciY);
   anima.vel = (pas) ? pas : 1;
   //Se asigna la posición inicial del elemento
   document.getElementById(capa).style.left = iniciX + "px";
   document.getElementById(capa).style.top = iniciY + "px";
   //Se calcula la longitud de la línea entre el inicio y el final de las coordenadas
   anima.cami = Math.sqrt((Math.pow((iniciX - finalX), 2)) + (Math.pow((iniciY - finalY), 2)));
   //Se calcula el tamaño en píxeles de los pasos a lo largo de los ejes
   anima.pasoX = parseInt(((anima.finX - anima.valorX) / anima.cami) * anima.vel);
   anima.pasoY = parseInt(((anima.finY - anima.valorY) / anima.cami) * anima.vel);
   //Se inicia la llamada respectiva a la animación
   anima.interval = setInterval("ejecutaAnimacion()", 10);
}
// Calcula los siguientes pasos y los asigna a las propiedades
function ejecutaAnimacion() {
   if ((anima.MoviX + anima.pasoX) <= anima.distX &&
   (anima.MoviY + anima.pasoY) <= anima.distY) {
      var x = anima.valorX + anima.pasoX;
      var y = anima.valorY + anima.pasoY;
      document.getElementById(anima.capa).style.left = x + "px";
      document.getElementById(anima.capa).style.top = y + "px";
      anima.MoviX += Math.abs(anima.pasoX);
      anima.MoviY += Math.abs(anima.pasoY);
      anima.valorX = x;
      anima.valorY = y;
   } else {
      document.getElementById(anima.capa).style.left = anima.finX + "px";
      document.getElementById(anima.capa).style.top = anima.finY + "px";
      clearInterval(anima.interval);
   }
```

```
}
```

La función anterior empieza con la creación del contenedor anima que se encarga de almacenar los valores para la animación. La llamada a la función animaRecta() asigna los valores iniciales del objeto anima y a continuación rellena sus propiedades con los valores pasados como parámetros o calculados a partir de éstos.

La función finaliza con la llamada al método setInterval(), que llama repetidas veces a la función que realmente implementa la animación ejecutaAnimacion(). Esta función mueve el objeto hacia su destino hasta que el elemento se acerca o llega a las coordenadas de destino; cuando no puede ir más allá, el elemento se posiciona explícitamente en el destino y el identificador del intervalo se limpia para detener las iteraciones.

3.6.2. Animación circular

El siguiente ejemplo se puede interpretar como una variante del anterior, ya que la diferencia es que en esta ocasión se va a animar el elemento haciendo que éste realice un trazado circular en lugar de rectilíneo.

Para ello, la nueva función animaCirc() va a necesitar de los siguiente parámetros:

- El identificador de la capa que se va a mover.
- La coordenada x del punto de inicio/final del círculo.
- La coordenada y del punto de inicio/final del círculo.
- Un valor entero par que indique el número de puntos que pintar en el círculo.
- Un valor entero del radio relativo del círculo.

Por ejemplo, la siguiente llamada de la función:

```
animaCirc("circula", 120, 120, 18, 5);
```

moverá el elemento desde el punto (120, 120), siguiendo una circunferencia de radio 5 píxeles.

```
// Se crea un objeto anima, que dispone de un conjunto
//de propiedades relacionadas con el movimiento
var anima = new Object();
//Se inicializa el objeto Anima
function initAnima() {
   anima = {capa:"", valorX:0, valorY:0, finX:0, finY:0, paso:1, puntos:1, radio:1,
   interval:null };
}
//Se completa el objeto anima con los parámetros y valores calculados necesarios
```

```
    function animaCirc(capa, iniciX, iniciY, puntos, radio) {
    initAnima();
    anima.capa = capa;
    anima.finX = anima.valorX = iniciX;
    anima.finY = anima.valorY = iniciY;
    anima.puntos = puntos;
    anima.radio = radio;
    //Asigna la posición inicial de los elementos
    document.getElementById(capa).style.left = iniciX + "px";
    document.getElementById(capa).style.top = iniciY + "px";
    //Comienza la repetición de la animación
    anima.interval = setInterval("ejecutaAnimacionC()", 10);
}
    function ejecutaAnimacionC() {
    if (anima.paso < anima.puntos) {
    var x = anima.finX +
    Math.round(Math.cos(anima.paso * (Math.PI/(anima.puntos/2)))
    * anima.radio);
    var y = anima.finY +
    Math.round(Math.sin(anima.paso * (Math.PI/(anima.puntos/2)))
    * anima.radio);
    document.getElementById(anima.capa).style.left = x + "px";
    document.getElementById(anima.capa).style.top = y + "px";
    anima.finX = x;
    anima.finY = y;
    anima.paso++;
    } else {
    document.getElementById(anima.capa).style.left =
    anima.valorX + "px";
    document.getElementById(anima.capa).style.top =
    anima.valorY + "px";
    clearInterval(anima.interval);
    }
}
```

El código de la animación circular es similar al que se planteó en el anterior ejemplo; éste se basa en la suavidad del movimiento circular, que se fundamenta en el número de puntos a lo largo del círculo, que se convierte en el límite superior de anima.paso en el if de la función ejecutaAnimacionC().

La matemática aplicada se basa en que se ha dividido el valor por PI para conseguir un círculo completo; de esta manera, para cualquier combinación de valores, el radio del círculo estará controlado por el factor de multiplicación que hay al final de las dos líneas que contienen Math.PI.

De hecho, el valor almacenado en anima.radio no es una medida recta en píxeles, sino un factor que controla el radio, de manera que, cuanto mayor sea el número, mayor es el radio.

Para concluir, cabe señalar que aumentar el valor de anima.puntos provoca que la animación sea más suave, ya que los arcos entre los puntos de refresco serán mayores, es decir, el movimiento será más suave, debido a que el tiempo del intervalo está fijado en 10 milisegundos y ello implica que se ejecute más rápido.

3.6.3. Crear un menú desplegable con JQuery

Siguiendo los siguientes pasos se creó un menú desplegable de dos niveles utilizando CSS y jQuery. La estructura HTML del menú será la siguiente:

```
<ul class="menu">
<li><a href="#">Inicio</a></li>
<li><a href="#">Tipo y requisitos</a>
<ul>
 <li><a href="#">Jubilaci&oacute;n</a></li>
 <li><a href="#">Pensi&oacute;n</a></li>
 <li><a href="#">Jubilaci&oacute;n por invalidez</a></li>
 <li><a href="#">Jubilaci&oacute;n por edad avanzada</a></li>
</ul>
</li>
<li><a href="#">Reajustes</a></li>
<li><a href="#">Contacto</a></li>
</ul>
```

Se utiliza CSS para aplicar formato. En primer lugar se sitúa cada elemento al lado del otro y no debajo del otro, tal como se sitúan por defecto las listas. Al elemento individual se le asigna el valor *position:relative*, para que, cuando una nueva lista parta del elemento se pueda posicionar correctamente utilizando *position:absolute*.

Al enlace se le aplica *text-transform:uppercase* para colocar en mayúsculas su texto, se define color de fondo y tipografía. Después se define el estado *:hover*, cambiando el color de fondo y de tipografía nuevamente para resaltarlo.

```
ul.menu {
   float:right;
   display:block;
   margin-top: 38px;
   list-style-type:none;
}
.menu li {
   line-height:18px;
```

```
    font-size:13px;
    position:relative;
    float:left;
}
.menu li a {
    color: #000;
    text-transform:uppercase;
    padding: 5px 20px;
    text-decoration:none;
}
.menu li a:hover {
    background: #9c0101;
    color: white;
}
```

En el siguiente paso se posiciona la nueva lista que se abre como hija de un elemento <li>. Por defecto esta oculta *display:none* y con position y top se acomoda la lista para que se despliegue hacia abajo.

```
.menu li ul {
    display:none;
    position:absolute;
    top:20px;
    width: 240px;
    background-color: #f4f4f4;
    padding:0;
    list-style-type:none;
}
```

Como colofón, se estilizan un poco los elementos hijos de la nueva lista.

```
.menu li ul li {
    width: 200px;
    border: 1px solid #9c0101;
    border-top:none;
    padding: 10px 20px;
}
.menu li ul li:first-child {
    border-top: 1px solid #9c0101;
}
.menu li ul li a {
    width: 240px;
    margin: 0;
    padding:0;
}
.menu li ul li a:hover {
    width: 240px;
```

```
    margin: 0;
    color: #9c0101;
    background:none;
}
```

Para acabar el menú se utiliza la librería jQuery, ya que simplifica el proceso. Con poco código se hace que cada ul li que tenga una etiqueta ul hija, se le modifique la propiedad display cada vez que el puntero del ratón pasa por arriba y sale del mismo. Ahora bien, para entender el código a continuación se explican unos detalles básicos de la librería.

Introducción jQuery:

Al igual que se han creado librerías a lo largo del curso, jQuery es una librería JavaScript que simplifica el tratamiento de documentos HTML, el manejo de eventos, la creación de animaciones y las interacciones vía Ajax. Además es multiplataforma, de manera que está testada en los siguientes navegadores: Internet Explorer 6.0+, FireFox 2.0+, Safari 2.0+, Opera 9.0+ y Chrome.

La última versión de la librería se puede descargar de la web:

http://docs.jquery.com/Downloading_jQuery#Current_Release,

de forma que se hace uso del fichero utilizando la sintaxis habitual:

```
<script type="text/javascript" src="jquery-1.7.2.min.js"></script>
```

A partir de este momento ya se pueden utilizar las funciones jQuery. Las funciones básicas se presentan a continuación:

La función central de jQuery es la función dolar $("") con la cual se accede a los nodos del árbol DOM.

A continuación unos sencillos ejemplos de uso:

```
$("div"); // obtiene todos los nodos DIV del árbol DOM
$("#layout"); // obtiene el nodo con el atributo id = "layout" del árbol DOM
$(".myStyle"); // obtiene los nodos de los atributos class que tengan el valor "myStyle" dentro
del árbol DOM
$("<li>home</li>"); // crea un nodo de tipo li con un texto embebido "home"
$("div#content"); // obtiene el nodo de tipo div cuyo identificador sea "content"
$("<li>home</li>").wrapInner("<a>"); // crea un nodo li e inserta un ancla
$("div#content").text("Este es el nuevo contenido de la capa"); // obtiene el nodo div con id =
content y modifica el texto que contiene la capa
```

La función $("") admite diferentes parámetros:

```
$("li"); //Selectores CSS
$(document);// Elementos DOM
$("<li>home<li>"); // HTML
$(function(){});//Funciones
```

Además dispone de métodos para asignar controlador de eventos:

- clic() i dbclick(): añade un evento al clic del ratón
- keypress() y keydown(): añade un evento a la pulsación del teclado
- hover(): añade dos eventos a la acción de pasar el puntero del ratón sobre un componente, siendo el primero al pasar por encima y el segundo al dejarlo
- toogle(): intercambia entre un comportamiento u otro cuando se produce el evento

También incorpora métodos para modificar los atributos asociados a los componentes o nodos del árbol DOM:

- text() i html(): retornan o asignan el texto como un *string* o html como si trabajásemos con innerHTML

- attr() i removeAttr(): accede, modifica y elimina un atributo de un nodo

- addClass(), removeClass() i toogleClass(): añade, elimina o intercambia las clases css

- css(): retorna o asigna una propiedad css a un nodo

La combinación de los dos tipos de funciones ejecuta código de una cierta complejidad como se observa en los siguientes ejemplos:

```
<script type="text/javascript">
// la función ready asociada al documento  asegura que el código se ejecuta
//cuando la página está totalmente cargada en el cliente
$(document).ready(function() {
//se obtienen todos los enlaces y se asocia una función al evento onclick
$("a").clic(function(event){
// cuando se produce el evento se asigna un estilo css de color de fondo en el enlace
$(this).css({backgroundColor:'red'});
});
});
</script>
```

Por lo tanto, el código restante del menú desplegable es el siguiente:

```
<script type="text/javascript" src="jquery-1.7.2.min.js"></script>
<script type="text/javascript">
```

```
    $(document).ready(function() {  //se ejecuta la función cuando la página está
     //cargada
    $('ul li:has(ul)').hover(  //se asigna en primer lugar el controlador de //el evento al pasar
    el puntero del ratón por la opción del menú
    function(i)
    {
    $(this).find('ul').css({display: "block"}); //se muestra al opción del menú
    },
    //la siguiente función se ejecutará cuando el puntero del ratón abandona el enlace
    function(i)
    {
    $(this).find('ul').css({display: "none"}); //se esconde la opción del menú
    }
    );
    });
</script>
```

3.7. Manipulación dinámica de las hojas de estilo

El W3C define CSS como "un mecanismo simple para añadir estilo (por ejemplo, fuentes, colores, espacios) en documentos Web". Por lo tanto, las hojas de estilo CSS (*Cascading Style Sheets*) son un mecanismo para aplicar formato a los documentos HTML (y a documentos en otros lenguajes estructurados, como XML), separando el contenido de las páginas de su apariencia.

La información está contenida en el documento HTML, pero en este documento no se especifica cómo debe ser visualizada o representada la información. Las indicaciones acerca de la presentación visual del documento estarán especificadas en las CSS.

El W3C, como no podía ser de otra manera, es el organismo que dicta también los estándares sobre las CSS. Se empezó a discutir en 1995 sobre la utilización de las hojas de estilo CSS para incluirlas en las especificaciones 3.0 del HTML, pero las discusiones se extendieron tanto que al final se impusieron las extensiones propuestas por Netscape, lo que dio lugar a la versión 3.2 del HTML, aunque la versión 3.0 del Explorer ya las soportaba en cierta medida.

En la actualidad, las especificaciones CSS se encuentran en el nivel 2 (CSS2).

3.7.1. Motivos para usar CSS

Son muchos los motivos que llevan al programador de espacios web al uso de hojas de estilo. A continuación, se enumeran los más destacados:

- CSS es un lenguaje estándar de diseño para la Web y, por lo tanto, aplicable a otros formatos como XML. Controla los colores, la tipografía, el tamaño y la ubicación de elementos e imágenes.

- CSS es muy sencillo de programar, por lo que no es imprescindible el uso de editores especializados.

- CSS permite un ahorro de ancho de banda, ya que un solo archivo CSS puede definir la estética de toda una web compleja.

- CSS reduce las operaciones de actualización y mantenimiento, dado que los encargados de contenidos no deberán preocuparse del aspecto final. Además, los cambios globales se pueden realizar en minutos: sólo con modificar un archivo CSS se modifica el aspecto de toda la web.

Si además se tiene en cuenta la convergencia por parte de los navegadores actuales en el cumplimiento del estándar, este uso se hace más interesante.

3.7.2. Cambiar una hoja de estilo

En este subapartado se va a mostrar un conjunto de acciones muy comunes relacionadas con la manipulación de CSS desde el lenguaje JavaScript. Éstas se presentarán en un ejemplo muy sencillo.

En ciertas web se permite que los usuarios definan un "skin", que no es más que el aspecto que tiene la web. Además, se puede implementar a partir de carga de una nueva página CSS cuando el usuario seleccione el nuevo aspecto.

A continuación, se plantea un ejemplo de esta técnica. Para empezar, se dispone de esta línea que define la hoja CSS por defecto de la web:

```
<link id="estiloPorDefecto" rel="stylesheet" type="text/css" href="estilo.css" />
```

La carga de una nueva hoja de estilos externa se puede realizar utilizando la siguiente sintaxis:

```
document.getElementById("estiloPorDefecto").href="estiloNuevo.css";
```

Por otra parte, es posible activar o desactivar una hoja de estilo de la página; para ello se utiliza la propiedad disabled del objeto StyleSheet:

```
document.styleSheets[1].disabled = true;
```

La activación es tan simple como la asignación del valor false a la propiedad anterior.

3.7.3. Leer valores de las propiedades de la hoja

A continuación, vamos a definir una función que averigua el valor de una propiedad de la hoja de estilo establecida mediante una etiqueta `<style>` o una hoja de estilo importada.

```
function getElementStyle(elemID, IEStyleProp, CSSStyleProp) {
   var elem = document.getElementById(elemID);
   if (elem.currentStyle) {
      return elem.currentStyle[IEStyleProp];
   } else if (window.getComputedStyle) {
      var compStyle = window.getComputedStyle(elem, ");
      return compStyle.getPropertyValue(CSSStyleProp);
   }
   return "";
}
```

La función devuelve el valor de la propiedad, bien sea definida en IE (utilizando el objeto currentStyle), bien en navegadores CSS (utilizando el método window.getComputedStyle() del W3C DOM).

Actividades

Seleccionad 5 ejemplos de la siguiente página web:

http://www.htmlpoint.com/dhtml/index.html

Implementadlos y comentad en el foro los problemas con los que os habéis encontrado y las características interesantes que habéis descubierto.