Desarrollo Web en entorno Cliente: JAVASCRIPT

UNIDAD 11. DOM de W3C

UNIDAD 11. DOM de W3C

DOM

Como ya dijimos en el capítulo anterior, ésta es la especificación del objeto Document, considerada estándar por la World Wide Web Consortium (W3C).

Qué es el DOM

El DOM es una API para XML1, lo que sin siglas quiere decir que es una capa de programación intermedia que representa un documento y que nos permite modificarlo. Por decirlo de otra manera, **es una serie de funciones y procedimientos que nos permiten trabajar sobre un modelo abstracto de un documento, que sirve como medio de comunicación entre nuestro lenguaje de programación —en este caso JavaScript— y los contenidos del documento.**

El DOM es una interfaz independiente de cualquier lenguaje de programación, pero aquí vamos a centrarnos en su uso por medio de JavaScript.

11.1 INTRODUCCIÓN

DOM se parece a una página en forma de árbol, donde el objeto Document es el tronco y los subsiguientes son las ramas. Para llegar a cualquier parte de una página (sea una etiqueta, el atributo de una etiqueta o texto), "atravesamos el árbol" saltando del tronco a una rama, a una subrama y así sucesivamente hasta llegar a la parte del árbol con la que queremos trabajar (más tarde veremos que también hay atajos que se pueden emplear para referirse directamente a los objetos de la página).

Para entender como funciona DOM primero necesitaremos entender unos cuantos términos básicos. Lo más importante es el concepto de **nodo**, que es un término genérico que usa DOM para referirse a **cualquier objeto dentro de un documento**.

Desde el punto de vista del contenido de un documento, hay dos tipos de nodo:

- **Elemento**. Este tipo de nodo se refiere a una etiqueta HTML dentro de un documento. Si la etiqueta tiene también una etiqueta final, esa etiqueta final es considerada parte del mismo elemento.
- **Nodo de texto**. Este tipo de nodo se refiere al texto que reside dentro de un bloque de etiquetas (esto es, entre una etiqueta y su etiqueta de cierre).

Por ejemplo, considere el siguiente código HTML:

```
<p>Mi página inicial</p>
```

Aquí, el elemento es la etiqueta , y el nodo de texto es el texto: `Mi página inicial`.

Desde el punto de vista de la estructura del documento, hay tres tipos de nodo:

- **Nodo padre**. Éste es un nodo que contiene uno o más nodos distintos. Aquí hay tres cosas a tener en cuenta:
  - Las etiquetas HTML que crean un bloque de etiquetas explícitas (es decir, que tienen una etiqueta de apertura y de cierre) **son siempre nodos padres**. Por ejemplo y o <table> y </table> o <li> o <ol> ….

- Los elementos que no crean ninguna clase de bloque de etiquetas **no pueden ser nodos padre**. Por ejemplo
 e <img />.

- **Nodo hijo.** Es un nodo contenido dentro de otro. Cualquier elemento o nodo de texto que resida dentro de un bloque de etiqueta es un nodo hijo. En el ejemplo mostrado antes, el nodo hijo es el nodo de texto *Mi página inicial*. Fíjarse que, como los nodos de texto no pueden contener ningún otro tipo de nodo, siempre son nodos hijo.
- **Nodo hermano.** Es un nodo que está al mismo nivel que otro. Por ejemplo, en el código mostrado abajo, el elemento <i>, el nodo de texto *a* y el elemento <b> son hermanos dentro del nodo padre <p>.

```
<p>
   <i>Bienvenido</i> a <b>Mi página inicial</b>
</p>
```

DOM considera los atributos de etiquetas como nodos separados (llamados nodos de atributo), pero no considera que un nodo de atributo sea el hijo del elemento que lo contiene. Por ejemplo, en un vínculo el nodo de atributo *href* está dentro del elemento *<a>*, pero no es considerado como hijo de ese nodo. Esto se refleja en el hecho de que **los atributos se ven también como propiedades del elemento contenedor**, lo que en la práctica hace inútil la idea de ver un atributo como un tipo separado de nodo.

Listado 10.1. Documento HTML simple

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<p>Here's some text.</p>
<p>Here's more text.</p>
<p>Link to the <a href="http://www.w3.org">W3</a></p>
</body>
</html>
```

En la figura 10.1 aparece el HTML para el listado 10.1 visto en la estructura de árbol del DOM.

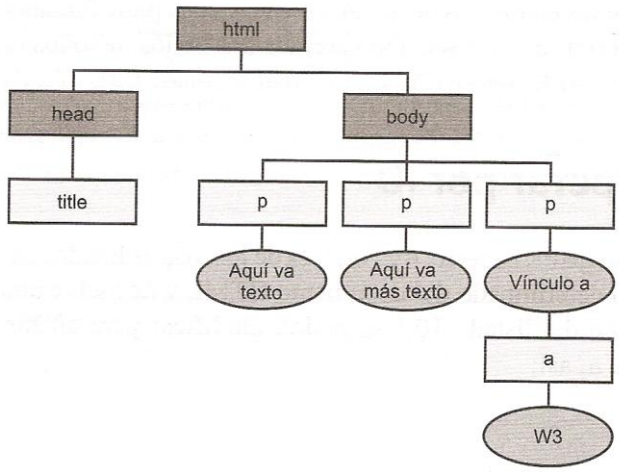
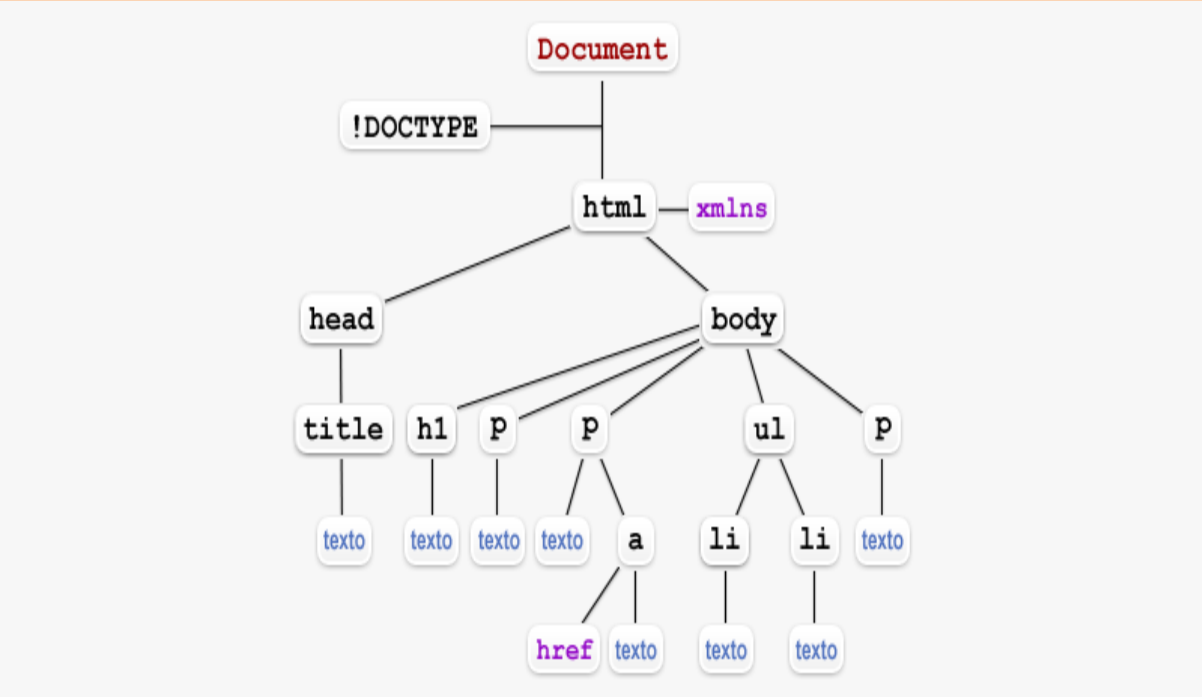


Figura 10.1. Documento simple representado como estructura de árbol.

Pongamos otro ejemplo. Tenemos un documento como el siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Título de la página</title>
</head>
<body>
    <h1>Encabezado</h1>
    <p>Un párrafo</p>
    <p>Un párrafo con <a href="otra_pagina.htm">un vínculo</a></p>
    <ul>
        <li>Un elemento de lista</li>
        <li>Otro elemento de lista</li>
    </ul>
    <p>Otro párrafo</p>
</body>
</html>
```

y por medio de JavaScript queremos modificar la lista. ¿Cómo lo hacemos? Necesitamos que haya alguna manera de indicarle al agente de usuario —por ejemplo, nuestro navegador— que de todos los caracteres que componen ese documento nos interesa seleccionar los que componen el que es un elemento ul. Ahí es donde entra en juego el DOM, que permite que el navegador se represente el documento como un árbol jerárquico, algo así:



El código anterior, representado gráficamente como un árbol jerárquico.

Ahora sí, el navegador no interpreta simplemente el documento como una serie de caracteres, sino que «comprende» que existen una serie de elementos con sus propiedades, y que la relación entre todos ellos es una relación estructural. Así, por ejemplo, se puede pedir al navegador que devuelva una lista de los hijos del elemento ul.

Una ventaja de esta interfaz es que la manera de trabajar con ella es análoga a la forma de emplear selectores de CSS basados en el árbol de parentesco que ya se ha estudiado.

El modelo resultante del documento original es un árbol, y una vez creado es este árbol lo que representa el navegador, y también lo que modificamos con nuestros *scripts*.

11.2 JERARQUÍA DE NODOS

Reiteramos aquí que **los nodos lo son todo en DOM**. Pasaremos la mayor parte del tiempo de codificación construyendo referencias para nodos individuales con el fin de trabajar con ellos. Esto significa que necesitamos entender cómo están estructurados los nodos en una página web.

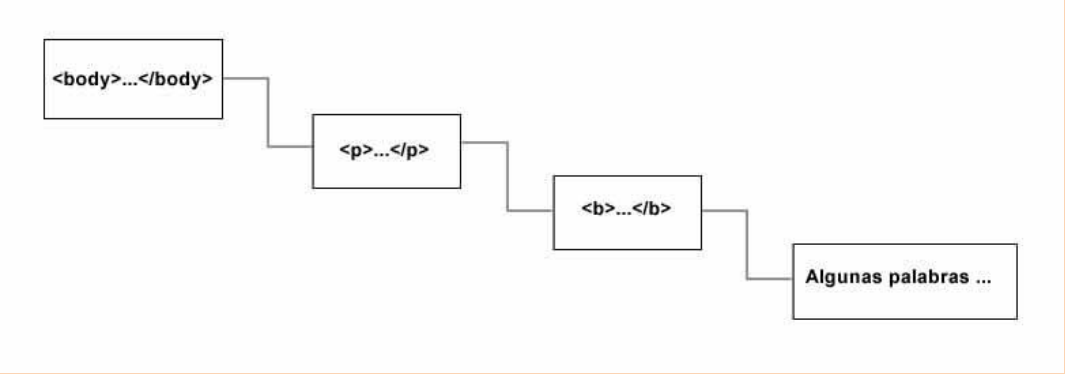
Lo primero que necesitamos saber es que los nodos de un documento son jerárquicos. El mismo objeto Document es un nodo, que se sitúa en la parte superior de la jerarquía. Observamos el siguiente código HTML.

```
<body>
  <p>
      <b>Algunas palabras en negrita</b>
  <p>
</body>
```

Esto es lo que tenemos:

- La etiqueta <p> es un nodo hijo del elemento <body>
- La etiqueta <b> es un nodo hijo del elemento <p>
- El nodo de texto *Algunas palabras en negrita* es un nodo hijo del elemento <b>

La siguiente figura muestra el aspecto de esta jerarquía.



Ahora observe este ejemplo ligeramente más complejo.

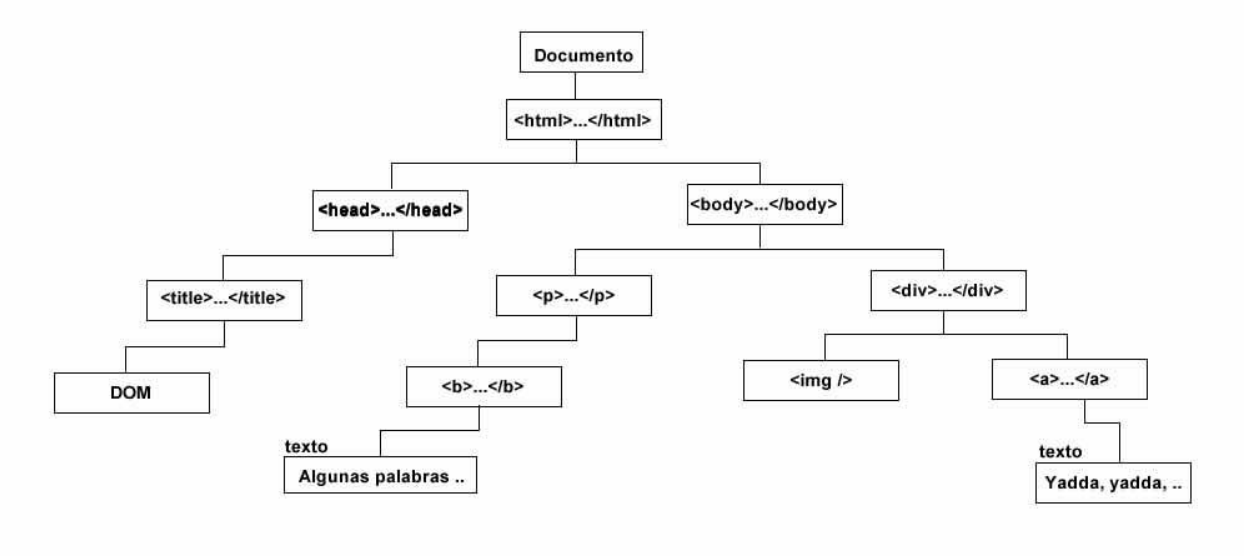
```
<html>
    <head>
       <title>DOM</title>
    </head>
    <body>
      <p>
        <b>Algunas palabras en negrita</b>
      </p>
```

```
      <div>
         <img src="mipic.gif" />
         <a href="yadda.html">Yadda, yadda, yadda</a>
      </div>
    </body>
</html>
```

Esto es lo que tenemos sobre este código actualizado:

- La etiqueta <html> es un nodo hijo del nodo *Document*
- Las etiquetas <head> y <body> son nodos hijo del elemento <html> y son nodos hermanos uno de otro.
- La etiqueta <p> es un nodo hijo del elemento <body>, como lo es la etiqueta . Esto convierte a la etiqueta <p> y a la etiqueta en hermanos.
- Las etiquetas <img /> y <a> son nodos hijo del elemento .
- El nodo de texto *Yadda, yadda, yadda* es un nodo hijo del elemento <a>

La figura siguiente muestra el aspecto de esta jerarquía.



Esta estructura jerárquica obliga a ver las páginas web bajo un nuevo punto de vista, ya que necesitará asegurarse de que los documentos están bien formados y de que tiene todos los nodos hijo dentro de los nodos padre adecuados. Esto le asegurará la manipulación satisfactoria de la estructura de la página después de que se haya cargado.

A continuación vamos a estudiar los elementos más importantes usados para manipular DOM.

11.3 ACCEDIENDO A LOS ELEMENTOS

Afortunadamente, Javascript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades.

Si deseamos **encontrar** de manera rápida y fácil **un elemento** se tiene a mano el **método getElementById**. El mismo **permite un acceso inmediato a cualquier elemento tan sólo conociendo el valor de su atributo id**. Véase el siguiente ejemplo:

```
<p>
    <a id="contacto" href="contactos.html">Contáctenos</a>
</p>
```

Puede usarse el atributo id del elemento *<a>* para acceder al mismo:

```
var elementoContacto = document.getElementById("contacto");
```

Ahora el valor de la variable elementoContacto está referida al elemento <a> y cualquier operación sobre la misma afectará el hiperenlace.

El **método getElementById es adecuado para operar sobre un elemento en específico**, sin embargo, en ocasiones se necesita **trabajar sobre un grupo de elementos** por lo que, en este caso podemos **utilizar el método getElementsByTagName**. Este **retorna todos los elementos de un mismo tipo**. Asumiendo la siguiente lista desordenada:

```
<ul>
    <li><a href="editorial.html">Editorial</a></li>
    <li><a href="semblanza.html">Autores</a></li>
    <li><a href="noticias.html">Noticias</a></li>
    <li><a href="contactos.html">Contáctenos</a></li>
</ul>
```

Puede obtenerse todos los hipervínculos de la siguiente manera:

```
var hipervinculos= document.getElementsByTagName("a");
```

El valor de la variable hipervínculos es una colección de elementos *<a>*. Las colecciones son arrays pudiéndose acceder a cada elemento a través de la ya conocida notación con corchetes.

Los elementos devueltos por getElementsByTagName serán ordenados según en el orden que aparezcan en el código fuente. Por tanto para el caso anterior quedaría así:

- hipervinculos[0] el elemento <a> para “Editorial”
- hipervinculos[1] el elemento <a> para “Autores”
- hipervinculos[2] el elemento <a> para “Noticias”
- hipervinculos[3] el elemento <a> para “Contáctenos”

Ejemplo anterior: **AccediendoElementos.html**

Como ya hemos visto en otra parte del tutorial existen varios elementos en un documento HTML que pueden ser accedidos de otras maneras. El elemento body de un documento puede accederse a través de la forma document.body, mientras que el conjunto de todos los formularios en un documento puede encontrase en document.forms, así mismo el conjunto de todas las imágenes sería mediante document.images.

Actualmente la mayoría de los navegadores soportan estos métodos aún así **es recomendable el uso del método getElementsByTagName**, véase el siguiente ejemplo para acceder al elemento body:

```
var cuerpo_body = document.getElementsByTagName("body")[0];
```

11.4 CREANDO ELEMENTOS Y TEXTO

La creación de nodos es posible mediante el uso de dos métodos disponibles en el objeto document. Dichos métodos son:

- createElement(Tipo cadena): Crea un nuevo elemento del tipo especificado y devuelve una referencia a dicho elemento.
- createTextNode(Cadena de texto): Crea un nuevo nodo de texto con el contenido especificado en la cadena de texto.

El siguiente ejemplo muestra cómo se crea un nuevo elemento de hiperenlace vacío:

```
var nuevoEnlace = document.createElement("a");
```

La variable nuevoEnlace ahora referencia un nuevo enlace listo para ser insertado en el documento. El texto que va dentro del elemento <a> es un nodo de texto hijo, por lo que debe ser creado por separado.

```
var nodoTexto = document.createTextNode("Semblanza");
```

Luego si deseamos **modificar el nodo de texto ya existente**, podemos utilizar la **propiedad nodeValue**, esta permite coger y poner el nodo de texto:

```
var textoViejo = nodoTexto.nodeValue;
nodoTexto.nodeValue = "Novedades";
```

El valor de la variable textoViejo es ahora "Semblanza" y el nuevo texto "Novedades". **Se puede insertar un elemento o texto (nodo) como último hijo de un nodo ya existente** usando el **método appendChild**. Este método coloca el nuevo nodo después de todos los hijos del nodo:

```
nuevoEnlace.appendChild(nodoTexto);
```

Ahora todo lo que necesitamos es insertar el enlace en el cuerpo del documento. Para hacer esto, **se necesita una referencia al elemento body del documento**, teniendo como guía los estándares siguientes:

```
var cuerpoRef = document.getElementsByTagName("body")[0];
cuerpoRef.appendChild(nuevoEnlace);
```

Otra manera sería utilizando el método getElementById. Para ello se asume que la etiqueta <body> tiene asignado un valor para el atributo id.

```
<body id=”cuerpo”>
var cuerpoRef = document.getElementById("cuerpo");
cuerpoRef.appendChild(nuevoEnlace);
```

Ejemplo anterior: **CreandoElementosTexto.html**

Existen básicamente **tres maneras mediante las cuales un nuevo elemento o nodo de texto puede ser insertado en una página Web**. Todo ello depende del punto en el cual se desee insertar el nuevo nodo: como **(1)último hijo de un elemento**, **(2)antes de otro nodo** o **(3)reemplazar un nodo**.

El caso de **(1)** apertura de un nuevo hijo ya fue visto en el ejemplo anterior, luego **(2) para insertar el nodo antes de otro nodo se realiza utilizando el método insertBefore de su elemento padre**, mientras que el **(3) reemplazo de nodo se utiliza el método replaceChild de su elemento padre**.

(2) Al **usar insertBefore**, se necesita tener referencias al nodo que va a ser insertado y donde va a ser insertado, considérese el siguiente código HTML:

```
<p id="mwEnlaces">
   <a id="editor" href="editorial.html">Editorial</a>
</p>
```

Luego el nuevo enlace será insertado antes del enlace ya existente llamando al método insertBefore desde el nodo padre (párrafo):

```
var nuevoAncla = document.createElement("a");
var anclaTexto = document.createTextNode("Actualidad");
nuevoAncla.appendChild(anclaTexto);

var anclaExistente = document.getElementById("editor");
var padre = anclaExistente.parentNode;
var nuevoHijo = padre.insertBefore(nuevoAncla, anclaExistente);
```

Si se hiciera una traducción del DOM hacia HTML después de esta operación el resultado sería el siguiente:

```
<p id="mwEnlaces">
   <a> Actualidad </a>
   <a id="editor" href="editorial.html">Editorial</a>
</p>
```

(3) En el **caso de reemplazar el enlace usando replaceChild**:

```
var nuevoHijo = padre.replaceChild(nuevoAncla, anclaExistente);
```

El DOM lucirá así:

```
<p id="mwEnlaces">
   <a> Actualidad </a>
</p>
```

Ejemplos anteriores: El alumno probará los casos **(2) insertBefore** y **(3) replaceChild**, ya que el **(1) apertura de un nuevo hijo** ya está hecho.

Usando innerHTML

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso recurriéndose a la **propiedad innerHTML**. Dicha propiedad fue introducida por Microsoft **permitiendo leer y escribir el contenido HTML de un elemento**.

Por ejemplo, puede crearse fácilmente una tabla con múltiples celdas e insertarla luego en la página con innerHTML:

```
var tabla = '<table border="0">';
tabla += '<tr><td>Celda 1</td><td>Celda 2</td><td> Celda 3</td></tr>';
tabla += '</table>';
document.getElementById("datos").innerHTML = tabla;
```

11.5 ELIMINANDO UN ELEMENTO O UN NODO DE TEXTO

Se pueden eliminar nodos existentes y nuevos. El **método removeChild** permite eliminar nodos hijos a cualquier nodo con tan sólo pasarle las referencias del nodo hijo a eliminar y su correspondiente padre. Para esto retomamos el ejemplo anterior:

```
<p id="mwEnlaces">
   <a id="editor" href="editorial.html">Editorial</a>
</p>
```

El método removeChild será usado para eliminar el hipervínculo del elemento padre párrafo:

```
var ancla = document.getElementById("editor");
var padre = ancla.parentNode;
var hijoRemovido = padre.removeChild(ancla);
```

La variable hijoRemovido todavía hace referencia al elemento, de manera que fue removido pero no destruido, no pudiéndose localizar en ninguna parte del DOM. Este se encuentra disponible en memoria como si fuera creado usando el método createElement.

Esto permite posicionarlo en cualquier otra parte de la página.

Ejemplo anterior: El alumno probará este último ejemplo

11.6 TRABAJANDO CON ATRIBUTOS (Lectura y escritura de un elemento)

Las partes más frecuentemente usadas de un elemento HTML son sus atributos, tales como: id, class, href, title, estilos CSS, entre muchas otras piezas de información que pueden ser incluidas en una etiqueta HTML.

Los atributos de una etiqueta son traducidos por el navegador en propiedades de un objeto. Existen dos **métodos para leer y escribir los atributos de un elemento, getAttribute permite leer el valor de un atributo mientras que setAttribute permite su escritura.**

En ocasiones se hace necesario ver las propiedades y métodos de un determinado elemento, esto puede realizarse mediante la siguiente función:

```
function inspector(el) {
   var str =””;
   for (var i in el){
      str+=I + “: ” + el.getAttribute(i) + “\n”;
   }
   alert(str);
}
```

Para usar la función inspector() tan sólo debemos pasarle la referencia al elemento, continuando con el ejemplo anterior resulta:

```
var ancla = document.getElementById("editor");
inspector(ancla);
```

Ejemplo anterior: **TrabajandoconAtributos.html**

Para modificar el **atributo title** del hipervínculo, o sea, el elemento referenciado por la variable ancla, se usará el **setAttribute**, pasándole el nombre del atributo y el valor:

```
var ancla = document.getElementById("editor");
ancla.setAttribute("title", "Artículos de programación");
var nuevoTitulo = ancla.getAttribute("title");
```

El valor de la variable nuevoTitulo es ahora `“Artículos de programación”.`

Ejemplo anterior: **TrabajandoconAtributos2.html**

11.7 LOS ESTILOS DE LOS ELEMENTOS (Manipulando los estilos de los elementos)

Como se ha visto, los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.

Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS pero sin los guiones y llevando la primera letra de las palabras a mayúsculas. Véase el siguiente ejemplo para mayor entendimiento donde se utiliza un atributo CSS modelo:

```
algun-atributo-css
```

Tendrá como equivalente la siguiente propiedad o método en Javascript:

```
algunAtributoCss
```

Por tanto, para cambiar el atributo CSS font-family de un elemento, podría realizarse lo siguiente:

```
ancla.style.fontFamily = 'sans-serif';
```

Los valores CSS en Javascript serán en su mayoría del tipo cadena; por ejemplo: font-size, pues posee dimensiones tales como "px", "%". Sólo los atributos completamente numéricos, tales como z-index serán del tipo entero.

En muchos casos es necesario hacer aparecer y desaparecer un determinado elemento, para ellos se utiliza el atributo CSS display, por ejemplo, para desaparecer:

```
ancla.style.display = 'none';
```

Luego para volverlo a mostrar se le asigna otro valor:

```
ancla.style.display = 'inline';
```

11.8 EJEMPLOS

Adjuntar múltiples ficheros a la vez

Este es el primer ejemplo completo donde se propone utilizar la manipulación del DOM mediante javascript con el objetivo de adicionar tantos elementos input del tipo file como tantos ficheros se deseen subir al servidor.

Se muestra una versión simplificada del problema limitada tan sólo al lado del cliente. Para ello es necesario imaginarse un sistema en línea donde se suben ficheros al servidor, ejemplo de ello podría ser una aplicación de correo electrónico.

```
 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
     <title>Ejemplo para adjuntar múltiples ficheros</title>
```

```
      <script language="javascript" type="text/javascript">
         function nuevoFichero() {
            var var_input = document.getElementsByTagName("input")[0];
            var nuevoInput = var_input.cloneNode(true);
            var_input.parentNode.appendChild(nuevoInput);
         }
      </script>
    </head>
    <body>
     <form action="upload.php" method="post" enctype="multipart/form-data">
        <fieldset><legend>Adjuntar múltiples ficheros</legend>
           <input name="ficheros[]" type="file" size="60" >
        </fieldset>
        <a href="javascript: nuevoFichero();">Adjuntar otro fichero</a>
        <input name="Subir" type="submit" value="Adjuntar" >
     </form>
    </body>
</html>
```

El enlace para adjuntar otro fichero hace una llamada a la función nuevoFichero() realizándose las siguientes tareas en la misma:

Se accede al primer elemento input encontrado en el documento:

```
var var_input = document.getElementsByTagName("input")[0];
```

- Se crea un nuevo elemento input referenciado por la variable *nuevoInput* utilizando el **método cloneNode** resultando un elemento idéntico al primero:

```
var nuevoInput = var_input.cloneNode(true);
```

- Aquí se accede al nodo padre del elemento input mediante el **método parentNode** y a la vez se inserta un nuevo elemento hijo copia del primero por medio del **método appendChild**:

```
Var_input.parentNode.appendChild(nuevoInput);
```

En este ejemplo podría haberse usado el evento clic para la llamada a la función.

Ejemplo anterior (Adjuntar múltiples ficheros a la vez): **MultiplesFicheros.html**

Mostrador de diapositivas

Este ejemplo contempla un mostrador de diapositivas (en este caso imágenes). Se tienen dos enlaces, uno para mostrar la diapositiva siguiente y otro para mostrar la diapositiva anterior. Una posible aplicación práctica podría ser en una galería de fotos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```
  <head>
  <title>Ejemplo de un mostrador de diapositivas</title>
  <script type="text/javascript">
    var contador = 0;
    var diapositivas = [];
    function inicio() {
      var contenedor = document.getElementById("diapositivas");
      while (contenedor.childNodes.length > 0){
        if (contenedor.getElementsByTagName("img")[0]==
          contenedor.firstChild){
            diapositivas[diapositivas.length] =
              contenedor.removeChild(contenedor .firstChild);
        }
        else contenedor.removeChild(contenedor.firstChild);
      }
    }
    function adelante() {
      contador++;
      if (contador >= diapositivas.length) contador = 0;
      ponerImagen();
    }
    function atras() {
      contador--;
      if (contador < 0 ) contador = diapositivas.length - 1;
      ponerImagen();
    }
    function ponerImagen() {
      var contenedor = document.getElementById("diapositivas");
      if (contenedor.childNodes.length==0)
        contenedor.appendChild(diapositivas[contador]);
      else
        contenedor.replaceChild(diapositivas[contador],
          contenedor.childNodes[0]);
    }
  </script>
  </head>
  <body>
    <a href="javascript: atras();">Atrás</a>
    <span id="diapositivas">
      <img src="diapositiva1.jpg" alt="Diapositiva 1"
        height="100" width="200">
      <img src="diapositiva2.jpg" alt="Diapositiva 2"
        height="100" width="200">
      <img src="diapositiva3.jpg" alt="Diapositiva 3"
        height="100" width="200">
    </span>
    <a href="javascript: adelante();">Adelante</a>
  </body>
</html>
<script type="text/javascript">
  inicio();
  ponerImagen();
</script>
```

Este ejemplo se analizará de manera abreviada.

Se tienen dos variables globales, *contador* y *diapositivas*, la primera para indicar la diapositiva que se está mostrando actualmente y la última para almacenar el conjunto de diapositivas a mostrar.

La ***función inicio()*** remueve todos los elementos img y los almacena. Ver que todo nodo hijo del nodo con id diapositivas (ejemplo: salto de línea) es eliminado para evitar un mal funcionamiento posterior.

Por otro lado la ***función ponerImagen()*** se encarga de colocar la imagen apuntada por el contador en el contenedor de diapositivas. Mientras que las ***funciones atras() y adelante()*** se encargan de decrementar e incrementar el contador respectivamente.

Ejemplo anterior (Mostrador de diapositivas): **MostradorDiapositivas.html**

Para consultas sobre DOM:

http://www.w3schools.com/jsref/dom_obj_document.asp

http://www.w3schools.com/js/js_htmldom.asp