Tema 3

El Modelo de Objeto de Documento (DOM)

1. Introducción a DOM

DOM define un conjunto estándar de atributos y métodos que los intérpretes de Javascript exponen para facilitar el acceso al contenido de los documentos HTML y XML desde sus programas. Es decir DOM es una representación de todos los objetos y elementos de una página web o de un documento XML. En sentido estricto podemos realizar la siguiente clasificación de DOM:

- `Core DOM`: Define un conjunto estándar de objetos para cualquier documento estructurado.
- `XML DOM`: Define un conjunto estándar de objetos para los documentos XML
- `HTML DOM`: Define un conjunto estándar de objetos para los documentos HTML

En esta unidad vamos a centrar mas la atención en HTML DOM aunque casi todo lo que aquí se comente es fácilmente trasladable a XML DOM.

DOM permite, mediante una rigurosa estructura jerárquica, acceder y manipular directamente todos los elementos (párrafos, links, imágenes, etc.) que componen el documento a través del uso de javascript. Esto va a permitir que dinámicamente, mediante la ejecución de un script, se actualicen elementos de nuestros documentos web y también que se creen nuevos elementos y se eliminen otros.

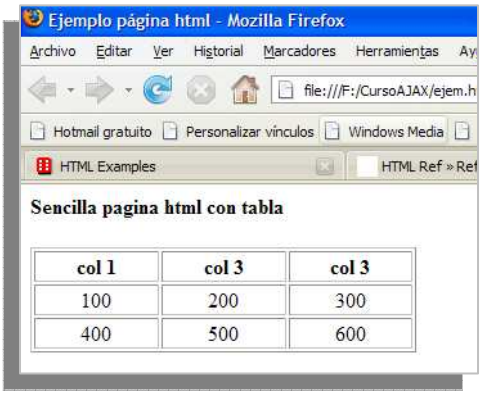
El uso de DOM es particularmente interesante para las aplicaciones AJAX. En las aplicaciones clásicas web lo que se hace es refrescar la página completa desde el servidor, en las aplicaciones AJAX, por el contrario, lo normal es modificar porciones del documento Web, no el documento Web completo y para ello es necesario hacer uso de la representación DOM de nuestra página.

2. Jerarquía de nodos DOM

Las etiquetas HTML en una página web están estructuradas en forma de árbol invertido. La raíz del árbol es la etiqueta inicial del documento <HTML>, a continuación y colgando directamente de la etiqueta raíz se encuentran las etiquetas <BODY> y <HEAD>. De la etiqueta <BODY> que representa la raíz de la parte visible del documento cuelgan las etiquetas que conforman párrafos, tablas, listas y el resto de tipos de etiquetas.

La representación DOM de una pagina web esta estructurada, como no podía ser de otra manera, de forma jerárquica en árbol y esta compuesta por elementos o *nodos,* cada elemento o nodo del documento se caracteriza porque solo puede tener un nodo padre y puede tener uno o más nodos hijos. Javascript trata cada elemento o nodo como un objeto compuesto por uno o más atributos y con métodos para su manipulación. Vamos a verlo con un ejemplo.

La siguiente imagen muestra una sencilla página web en el navegador



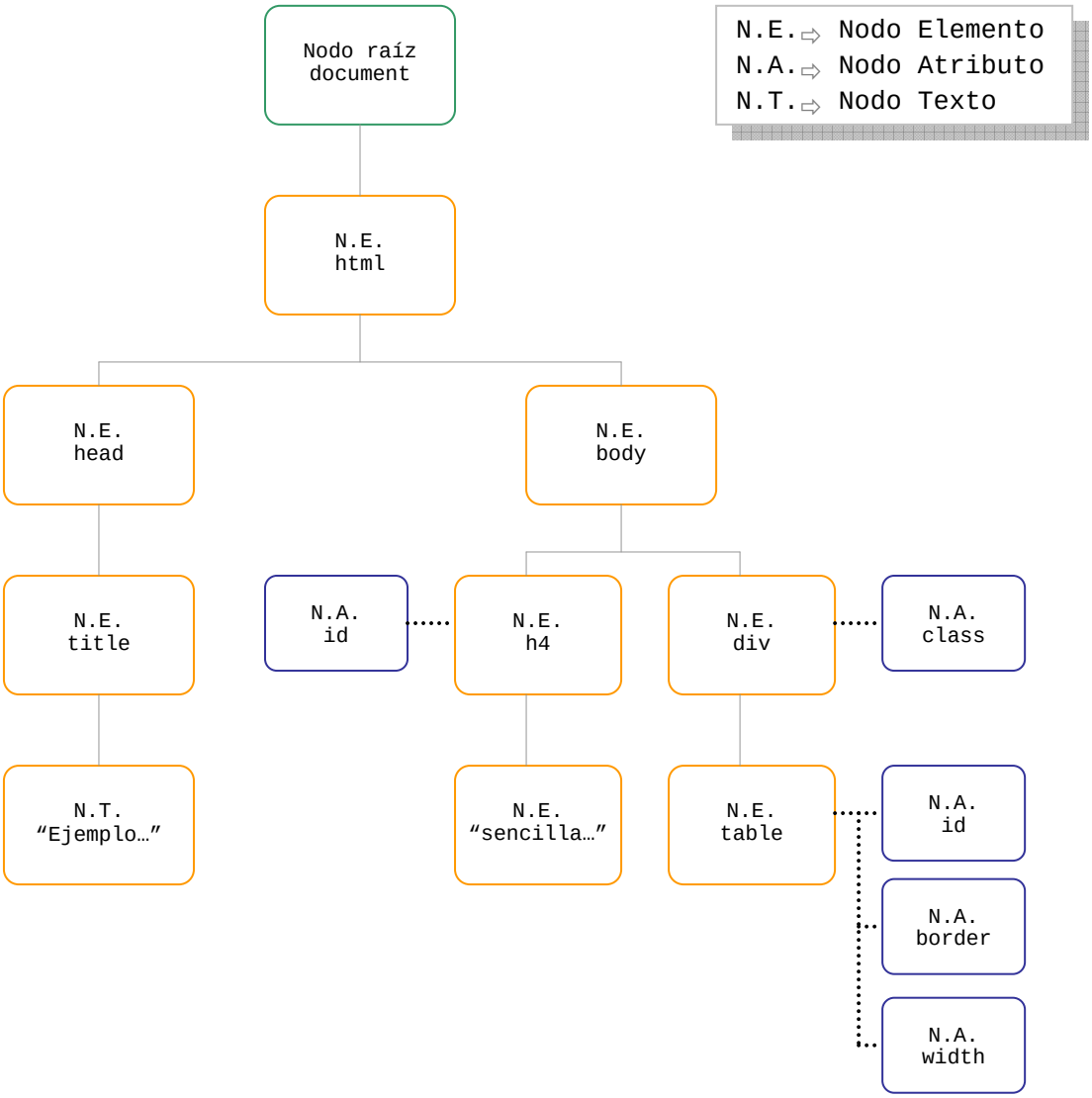
El código HTML es el que se muestra a continuación:

```
<html>
  <head>
    <title>Ejemplo página html</title>
  </head>
  <body>
<h4 id='titulo'>Sencilla pagina html con tabla</h4>
    <div class='principal'>
    <table id='mitabla' border='1' width='300' >
      <tr id='línea cabecera' >
        <th>col 1</th>
        <th>col 3</th>
        <th>col 3</th>
     </tr>
     <tr align='center' >
        <td>100</td>
        <td>200</td>
        <td>300</td>
     </tr>
     <tr align='center' >
```

```
        <td>400</td>
        <td>500</td>
        <td>600</td>
      </tr>
    </table>
    </div>
  </body>
</html>
```

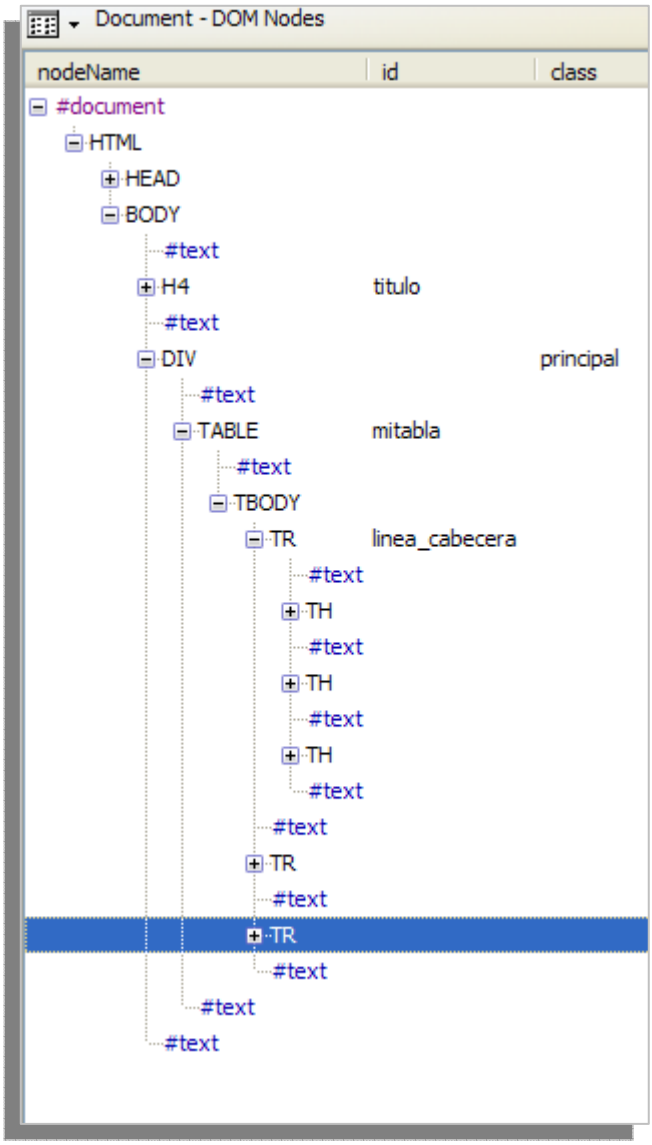
En este ejemplo vemos como algunas etiquetas HTML tienen un identificador `id` asociado que permite identificar de forma univoca la etiqueta referenciada. Ya se vio en el tema anterior sobre CSS la utilidad de los identificadores en las etiquetas para la aplicación de estilos. Los identificadores de etiquetas también permiten acceder a los nodos DOM de forma muy eficaz ya sea para modificar su contenido, eliminarlo, sustituirlo…

En DOM hay tres tipos de nodos (nodo elemento, nodo atributo y nodo texto) y todos ellos cuelgan del nodo raiz *document*. Los nodos elemento se corresponden a las etiquetas HTML del documento, los nodos atributo se corresponden a los atributos de dichas etiquetas y los nodos de texto son las cadenas de caracteres que se encuentran entre las etiquetas.



El Modelo de Objeto de Documento (DOM)

Vamos a ver la jerarquía DOM del código HTML mostrado anteriormente. Una forma muy cómoda de verlo, si se usa el navegador Firefox, es mediante la herramienta asociada a este navegador que se llama *DOM Inspector*.



En la imagen superior, aunque no están desplegados completamente todos los nodos, vemos como *DOM Inspector* muestra la jerarquía de árbol a partir del nodo raíz *document*. Con esta herramienta, a base de contraer y desplegar nodos, podemos centrarnos en la jerarquía de algún subnodo especial dentro de la página y también permite localizar fácilmente los atributos `id` y `class` asociados a las etiquetas. En la imagen se pueden observar muchos nodos # *text* que no llevan texto asociado que se visualice en la página eso se debe a que cada salto de página que se realiza en el código HTML, simplemente para hacer mas legible el código, genera un nodo de texto que solo lleva un salto de línea como cadena de texto asociada.

3. Buscar y acceder a los nodos

Antes de poder manipular el contenido o el estilo de cualquier nodo hay que saber acceder a el. Existen diversas formas de buscar y localizar nodos en un documento, lo más frecuentemente es utilizar, bien los métodos `getElementById()`o `getElementsByTagName()`, bien las cuatro propiedades que tiene todo nodo: `parentNode`, `firstChild`, `lastChild` y `childNodes.`

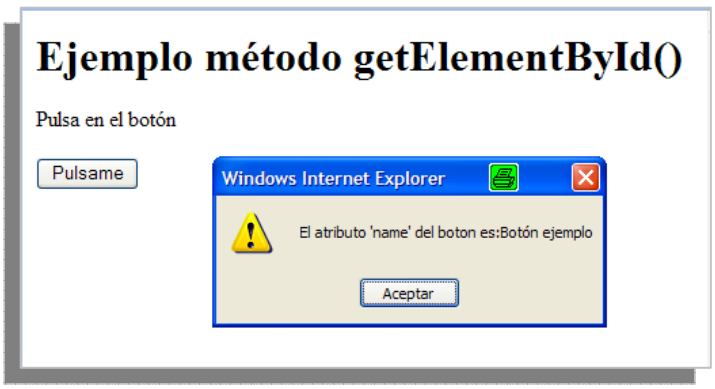
- **El método `getElementById()`**

El método `getElementById()`permite acceder a un nodo específico dentro del documento siempre que lleve un valor `id` asociado que permita su localización. Ya se comentó en el tema anterior que el atributo `id` de las etiquetas debe contener un valor único dentro del documento para que sea efectivo. En el caso de que dos atributos id tengan el mismo valor el método retornará el primero que localice.

Vamos a ver con un ejemplo de una sencilla página HTML el uso de `getElementById()`:

```
<html>
<head>
<script type="text/javascript">
function getName()
{
var x=document.getElementById("miboton")
alert("El atributo 'name' del boton es:"+x.name)
}
</script>
</head>
<body>
<h1 id="micabecera">Ejemplo método getElementById()</h1>
<p>Pulsa en el botón </p>
<input type="button" id="miboton"
value="Pulsame" name="Botón ejemplo" onclick="getName()">
</body>
</html>
```

La función Javascript `getName()` crea una variable x que va a guardar el objeto nodo retornado por el método `getElementById()`, este método, al ser un método de localización de nodo, siempre se aplica a todo el documento por eso se llama a este método desde el objeto *document* que engloba a todos los nodos. A continuación despliega una ventana de alerta para mostrar el atributo `name` del nodo cuyo `id` se asocia al valor `miboton`.



- **El método `getElementsByTagName()`**

Este método retorna, en un array de objetos, todos los nodos del documento que coinciden con el tipo de etiqueta que se le pasa al método como argumento.

A diferencia del método anterior que siempre se llamaba desde el objeto raíz *document*, este método se puede invocar desde cualquier nodo del documento. Es decir, la sintaxis puede ser:

```
document.getElementsByTagName("n_etiqueta");
```

y también,

```
document.getElementById('valorID').getElementsByTagName("n_etiqueta");
```

Es decir, la primera llamada retornará todos los nodos del documento que tengan la etiqueta HTML `n_etiqueta`, y la segunda llamada, retornará solamente, los nodos `n_etiqueta` que se encuentren por debajo jerárquicamente (nodos hijos) del nodo identificado como `valorID`.

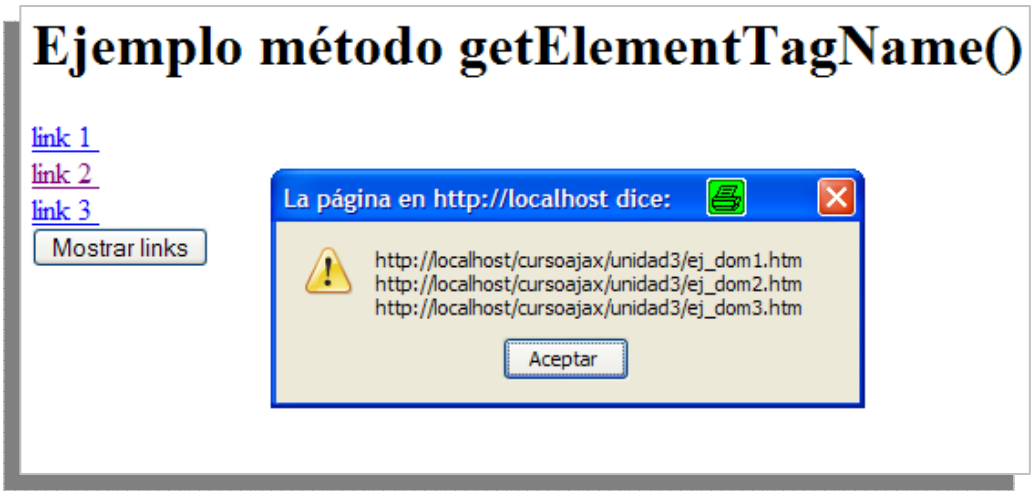
Veamos el siguiente ejemplo, donde aparece una página con una serie de links (etiqueta `<a>` de HTML) cómo, mediante el uso de la función `getElementsByTagName`, al pulsar el botón, se van a guardar todas las etiquetas `<a>` en un array `x`. Un bucle va a recorrer el array para extraer el atributo `href` y generar una cadena de salida que se visualiza mediante una ventana de alerta.

```
<html>
<head>
<script type="text/javascript">
function getLinks()
  {
  var salida="";
  var x=document.getElementsByTagName("a");
  for(i=0;i<x.length;i++)
  {
          salida+=x[i].href;
  salida+='\n'
  }
  alert(salida);
  }
</script>
</head>

<body>
<h1 >Ejemplo método getElementTagName()</h1>
<a href="ej_dom1.htm"> link 1 </a><br />
<a href="ej_dom2.htm"> link 2 </a><br />
<a href="ej_dom3.htm"> link 3 </a><br />

<input type="button" onclick="getLinks()" value="Mostrar links" />
</body>
</html>
```

En el código podemos ver cómo se determina el número de links de la página mediante `x.length` y cómo accedemos a cada uno de los atributos `href` de cada link: mediante el índice

del array `x[i].href`. La variable `salida` no es más que una cadena que se va construyendo al concatenar los distintos enlaces de la página a medida que se recorre el bucle `for`.

El resultado se observa en la imagen que se ve a continuación:



- **Las propiedades `parentNode`, `firstChild`, `lastChild` y `childNodes`**

Estas cuatro propiedades pertenecen a todo nodo del documento.

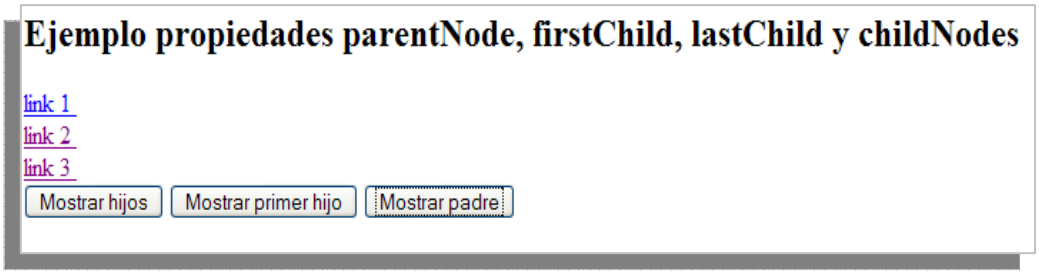
- `firstChild` y `lastChild` retornan el primer y último nodo hijo directo respectivamente de un nodo concreto o retorna `null` si no tiene nodos hijos.
- `parentNode` retorna el nodo padre de un nodo concreto o `null` si el nodo en cuestión no se encuentra dentro del documento DOM.
- `childNodes` retorna un array con todos los nodos hijos de un nodo concreto o `null` si no tiene nodos hijos.

Veamos el siguiente ejemplo, donde se muestra cómo actúa cada una de estas propiedades. En este ejemplo se crean las funciones `gethijos()`, `getprimerhijo()` y `getpadre()`, que sirven para mostrar el comportamiento de las propiedades `childNodes`, `firstChild` y `parentNode` respectivamente.

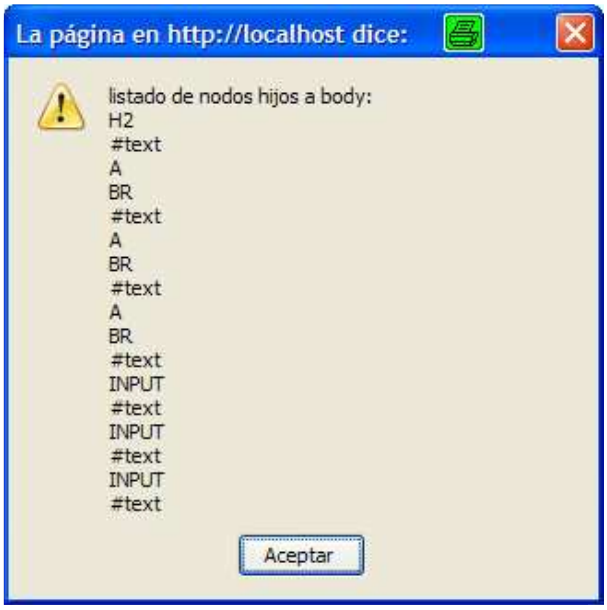
```
<html>
<head>
<script type="text/javascript">
function gethijos()
  {
  var salida="listado de nodos hijos a body:\n";
  var x=document.getElementById("inicio");
  var hijos=x.childNodes;
  for(i=0;i<hijos.length;i++)
  {       salida+=hijos[i].nodeName;
 salida+='\n'
  }
  alert(salida);
```

```
  }

  function getprimerhijo()
  {
  var salida="Primer hijo de body :\n";
  var x=document.getElementById("inicio");

          salida+=x.firstChild.nodeName;
  alert(salida);
  }
  function getpadre()
  {
  var salida="Padre de body :\n";
  var x=document.getElementById("inicio");
          salida+=x.parentNode.nodeName;
  alert(salida);
  }
</script>
</head>

< body id='inicio'><h2>Ejemplo propiedades parentNode, firstChild, lastChild y
childNodes
</h2>
<a href="ej_dom1.htm"> link 1 </a><br />
<a href="ej_dom2.htm"> link 2 </a><br />
<a href="ej_dom3.htm"> link 3 </a><br />

<input type="button" onclick="gethijos()" value="Mostrar hijos" />
<input type="button" onclick="getprimerhijo()" value="Mostrar primer hijo" />
<<input type="button" onclick="getpadre()" value="Mostrar padre" />
</body>
</html>
```

La página que se muestra al cargar el ejemplo de la página web anterior es la siguiente



Al pulsar el botón de *Mostrar hijos* se obtiene la siguiente información generada por la función javascript `gethijos()`. Si nos fijamos en el cuerpo de la función vemos que la variable `hijos` guarda en forma de array de nodos el retorno de la propiedad `childNodes` del nodo correspondiente a la etiqueta `body` de la página que tiene el `id` asociado con el valor `inicio`. Mediante un bucle `for` generamos la salida con los nombres de los nodos hijos que se visualiza en la siguiente ventana de alerta:



Al pulsar el botón *Mostrar primer hijo* lo que hacemos es llamar a la función `getprimerhijo()` que se limita a obtener el nombre del nodo del primer hijo mediante el uso de la propiedad `firstChild`

El botón *Mostrar padre*, a su vez, llama a la función `getpadre()` que usa la propiedad `parentNode` para visualizar el nombre del nodo padre a la etiqueta `body` que corresponde, en la jerarquía de nodos, a la etiqueta `HTML` del documento, tal como se ve al visualizar la ventana de alarma:

4. Crear, eliminar y modificar nodos DOM

- **Modificar el contenido de un nodo.**

Hay una propiedad muy útil para modificar el contenido de un nodo: `innerHTML`. Esta propiedad, asociada a un nodo cualquiera, establece o retorna, todo el contenido HTML que existe entre el principio y el final del nodo, incluyendo otras etiquetas HTML. Vamos a verlo en la siguiente muestra de código

```
<html>
<body>
<table id='mitabla' border='1' width='300' >
<tr id='linea_cabecera' >
  <th>col 1</th>
  <th>col 3</th>
  <th>col 3</th>
</tr>
</table>
<script type="text/javascript">
var x=document.getElementById("mitabla");
alert(x.innerHTML);
</script>
</body>
</html>
```

Esta página tiene un pequeño script que se ejecuta al cargar la página. Este script localiza el nodo con el `id` identificado como `mitabla` y muestra, en una alerta, el contenido de la propiedad `innerHTML` de este nodo, que equivale al código HTML de la página que se encuentra entre el principio y el final de dicho nodo, en la jerarquía de la página. Si cargamos la página vemos la siguiente imagen:

Si añadimos una fila al script para cambiar el valor de la propiedad `innerHTML` al nodo seleccionado de esta forma…

```
<script type="text/javascript">
var x=document.getElementById("mitabla");
x.innerHTML="<tr><td>Cambio tabla</td></tr>";
alert(x.innerHTML);
</script>
```

…al cargar de nuevo la página vemos como ha cambiado el código HTML interno al nodo `mitabla`

- **Crear nuevos nodos en la página**

Mediante el uso de los métodos DOM se puede fácilmente añadir nuevos elementos en una página web sin necesidad de refrescar completamente la página.

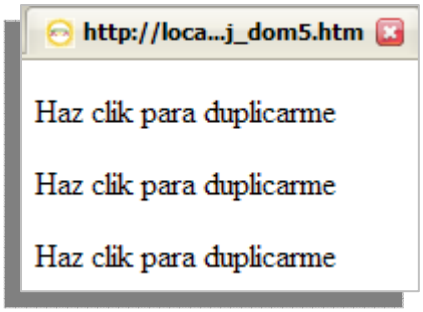
El método que permite crear nuevos nodos de tipo elemento es `document.createElement()` . A este método se le pasa la etiqueta HTML para definir el tipo de elemento que se va a crear. Una vez creado el elemento se pueden manipular sus atributos a conveniencia. Con el elemento creado y sus atributos ajustados, falta un último paso que es incorporarlo al documento. Una forma habitual de hacerlo es mediante el uso del método `appendChild()`una vez localizado el nodo padre del nuevo nodo elemento creado.

El siguiente ejemplo muestra una página con un nodo `div` que contiene inicialmente un único nodo <p> con un texto. Al pinchar con el ratón dentro del nodo `div` se ejecuta el script que crea un nuevo nodo <p> idéntico dentro del nodo `div` padre.

```
<html>
<head>
<script type="text/javascript">
function creanodo(){
  var nuevo=document.createElement('p');
  nuevo.innerHTML="Haz clik para duplicarme";
  var x=document.getElementById("texto");
  x.appendChild(nuevo);
}
</script>
</head>
```

```
<body>
<div id="texto" onclick="creanodo()">
<p>Haz clik para duplicarme</p>
</div>

</body>
</html>
```

El código es bastante simple. La variable `nuevo` almacena el nodo creado. Mediante `innerHTML` añadimos el contenido HTML del nodo, en este caso es el texto dentro del párrafo, y la variable `x` almacena el nodo padre (la etiqueta `div`) mientras que, mediante `appenChild()` añadimos el nodo creado como un nuevo hijo. Si pinchamos varias veces el ratón sobre el texto la página va añadiendo nodos párrafo como se muestra en la siguiente imagen:



- **Eliminar nodos en la página**

La forma mas usual de eliminar un nodo elemento de la página es determinar el nodo padre del nodo a eliminar y usar el método `removeChild()` . Un sencillo ejemplo es una página con un botón que, al ser pinchado, desaparece. El código es el siguiente:

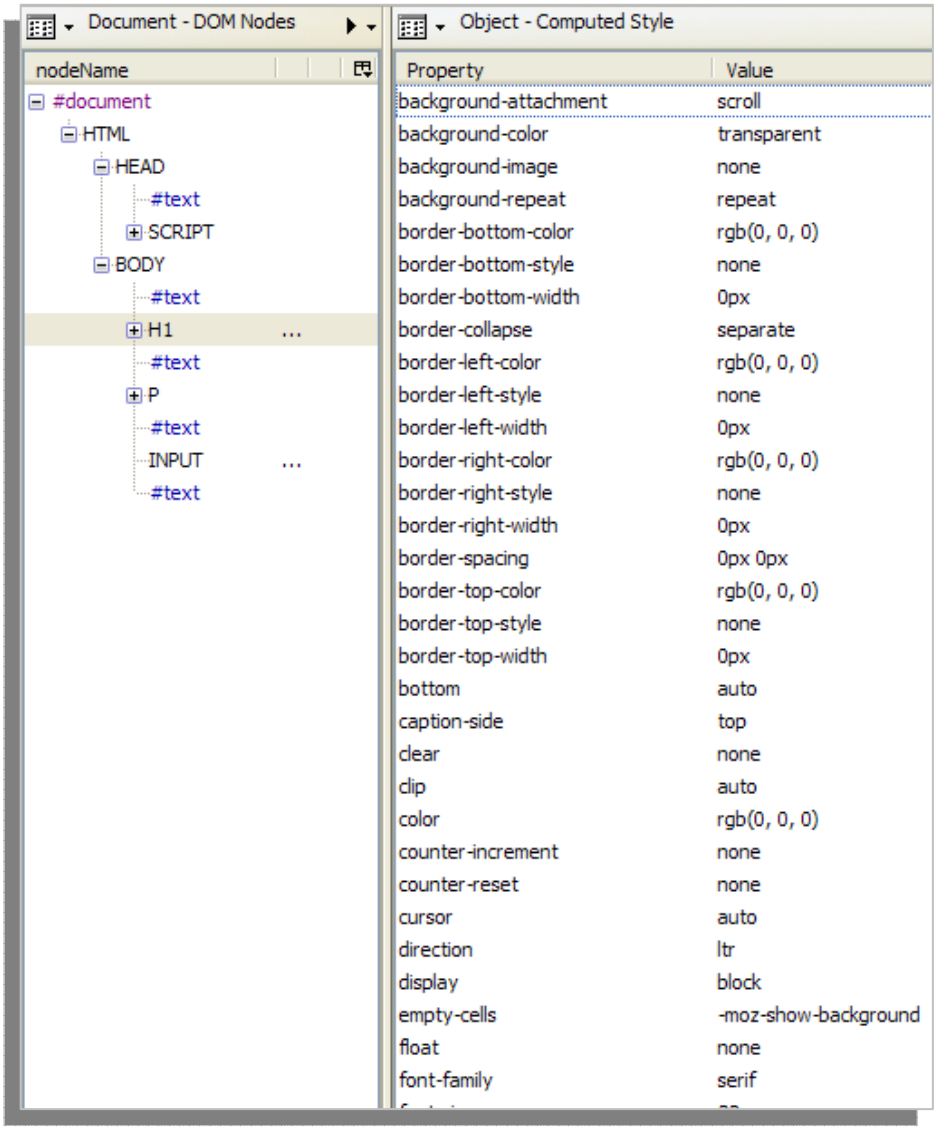
```
<html>
<head>
<script type="text/javascript">
function eliminanodo()
{
var x=document.getElementById("miboton");
x.parentNode.removeChild(x);
}
</script>
</head>
<body>

<h1 id="micabecera">Ejemplo método removeChild()</h1>
<p>Pulsa en el botón </p>
<input type="button" id="miboton"
value="Pulsame para eliminarme"  onclick="eliminanodo()">
</body>
</html>
```

5. Añadir estilos al documento.

Cada elemento de una página web tiene una serie de atributos visuales que pueden ser modificados dinámicamente a través de DOM, como puede ser la posición, color, anchura, márgenes, bordes, etc.

- **La propiedad `style`**

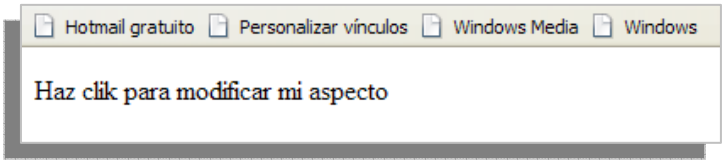
Cada nodo DOM contiene un array asociativo llamado `style` conteniendo todos y cada uno de los atributos de estilo que definen ese nodo. La siguiente figura muestra el array `style` visto por el programa *DOM Inspector* de un determinado nodo `<H1>` de una página web.



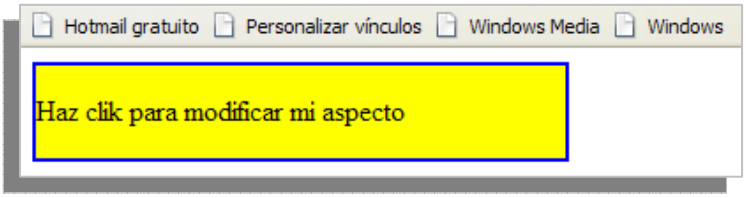
Este array puede ser modificado directamente accediendo de forma individual a cada uno de sus elementos. El siguiente ejemplo simple lo muestra:

```
<html>
<head>
<script type="text/javascript">
function cambianodo(){
  var x=document.getElementById("texto");
  x.style.border='solid blue 2px';
  x.style.width='300px';
  x.style.background='yellow'
}
</script>
</head>
<body>
<div id="texto" onclick="cambianodo()">
<p>Haz clik para modificar mi aspecto</p>
</div>



</body>
</html>
```

En este ejemplo se ve como pulsando con el ratón sobre la zona `<div>` identificada como `texto` modificamos sus atributos cambiando el borde, la anchura y el color de fondo.



Al pulsar se visualiza el cambio perseguido



- **La propiedad `className`**

Esta propiedad es muy interesante a la hora de cambiar o añadir estilos a elementos de una página web, ya que aprovecha las posibilidades de CSS y de este modo no es necesario acceder individualmente a los valores del array `style` para su modificación dinámica. Esta propiedad `className` permite asignar dinámicamente, a cualquier nodo, el atributo `class` del documento `css` asociado.

Si cambiamos el ejemplo anterior para trabajar con `className` vemos como cambia el código

```
<html>
<head>
<style type="text/css">
.cajatexto {background-color:yellow;
                  width:300px;
                  border:solid blue 2px}
</style>
<script type="text/javascript">
function cambianodo(){
  var x=document.getElementById("texto");
  x.className='cajatexto';

}
</script>
</head>
<body>
<div id="texto" onclick="cambianodo()">
<p>Haz clik para modificar mi aspecto</p>
</div>

</body>
</html>
```

En lugar de modificar directamente mediante `style`, a cada uno de los elementos del nodo identificado como `texto`, se le asigna la clase `css` identificada como `cajatexto`, que es la encargada de definir el nuevo estilo del nodo. Esta forma de modificar el estilo, mediante el uso de `className` es mucho mas recomendable, y eficiente, que la anterior, basada en modificar directamente el array `style` de un nodo concreto. No obstante, esta última opción puede ser de mucha utilidad en casos muy concretos y puntuales.