Listado de objetos del DOM en HTML:

Document	HTMLElement	Anchor	Area	Base
Body	Button	Event	Form	Frame/IFrame
Frameset	Image	Input Button	Input Checkbox	Input File
Input Hidden	Input Password	Input Radio	Input Reset	Input Submit
Input Text	Link	Meta	Object	Option
Select	Style	Table	TableCell	TableRow
Textarea				

Introducción al DOM. **http://www.librosweb.es/ajax/capitulo4.html**

"Por muy alto que sea un árbol, sus hojas siempre caen hacia la raíz."

Anónimo

1.2.- El árbol del DOM y tipos de nodos.

La tarea más habitual en programación web suele ser la manipulación de páginas web, para acceder a su contenido, crear nuevos elementos, hacer animaciones, modificar valores, etc.

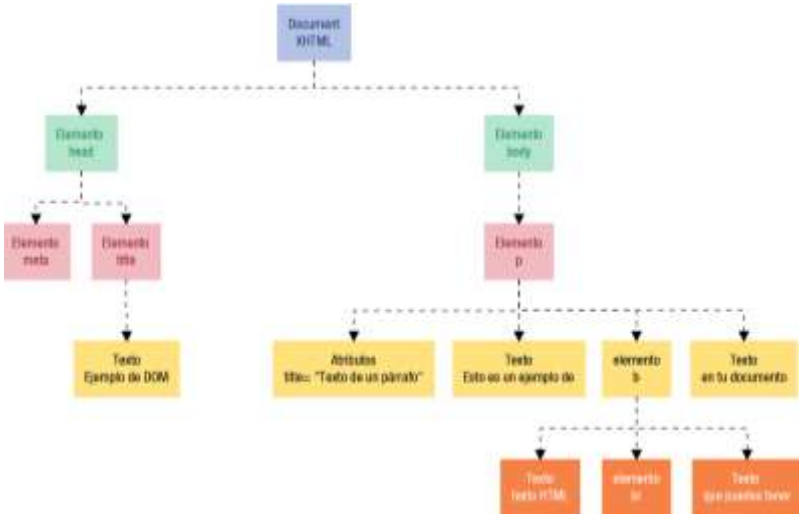
Todas estas tareas se pueden realizar de una forma más sencilla gracias al DOM. Los navegadores web son los encargados de realizar la transformación de nuestro documento, en una estructura jerárquica de objetos, para que podamos acceder con métodos más estructurados a su contenido.

El DOM transforma todos los documentos XHTML en un conjunto de elementos, a los que llama **nodos**. En el HTML DOM **cada nodo es un objeto**. Estos nodos están conectados entre sí y representan los contenidos de la página web, y la relación que hay entre ellos. Cuando unimos todos estos nodos de forma jerárquica, obtenemos una estructura similar a un árbol, por lo que muchas veces se suele referenciar como árbolDOM, "**árbol de nodos**", etc.

Veamos el siguiente ejemplo de código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1ransitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo de DOM</title>
   </head>
   <body>
      <p title="Texto de un párrafo">Esto es un ejemplo de <b>texto HTML<br />que puedes
tener</b> en tu documento.</p>
   </body>
</html>
```

Ese código se transformaría en el siguiente árbol de nodos:



Cada rectángulo del gráfico representa un nodo del DOM, y las líneas indican cómo se relacionan los nodos entre sí. La raíz del árbol de nodos es un nodo especial, denominado "**document**". A partir de ese nodo, cada etiqueta XHTML se transformará en nodos de tipo "**elemento**" o "**texto**". Los nodos de tipo "texto", contendrán el texto encerrado para esa etiqueta XHTML. Esta conversión se realiza de forma jerárquica. El nodo inmediatamente superior será el **nodo padre** y todos los nodos que están por debajo serán **nodos hijos**.

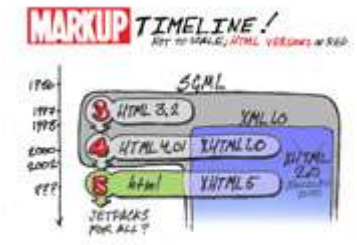
Tipos de nodos.

La especificación del DOM define 12 tipos de nodos, aunque generalmente nosotros emplearemos solamente cuatro o cinco tipos de nodos:

- ✓ `Document`, es el nodo raíz y del que derivan todos los demás nodos del árbol.
- ✓ `Element`, representa cada una de las etiquetas XHTML. Es el único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- ✓ `Attr`, con este tipo de nodos representamos los atributos de las etiquetas XHTML, es decir, un nodo por cada atributo=valor.
- ✓ `Text`, es el nodo que contiene el texto encerrado por una etiqueta XHTML.
- ✓ `Comment`, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos pueden ser: `CdataSection`, `DocumentFragment`, `DocumentType`, `EntityReference`, `Entity`, `Notation` y `ProcessingInstruction`.

1.3.- Acceso a los nodos.



Cuando ya se ha construido automáticamente el árbol de nodos del DOM, ya podemos comenzar a utilizar sus funciones para acceder a cualquier nodo del árbol. El acceder a un nodo del árbol, es lo equivalente a acceder a un trozo de la página de nuestro documento. Así que, una vez que hemos accedido a esa parte del documento, ya podemos modificar valores, crear y añadir nuevos elementos, moverlos de sitio, etc.

Para acceder a un nodo específico (elemento XHTML) lo podemos hacer empleando dos métodos: o bien a través de los nodos padre, o bien usando un método de acceso directo. A través de los nodos padre partiremos del nodo raíz e iremos accediendo a los nodos hijo, y así sucesivamente hasta llegar al elemento que deseemos. Y para el método de acceso directo, que por cierto es el método más utilizado, emplearemos funciones del DOM, que nos permiten ir directamente a un elemento sin tener que atravesar nodo a nodo.

Algo muy importante que tenemos que destacar es, que para que podamos acceder a todos los nodos de un árbol, el árbol tiene que estar completamente construido, es decir, cuando la página XHTML haya sido cargada por completo, en ese momento es cuando podremos acceder a cualquier elemento de dicha página.

Consideremos el siguiente ejemplo y veamos las formas de acceso:

```
<input type="text" id="apellidos" name="apellidos" />
```

getElementsByName()

Esta función obtiene una colección, que contiene todos los elementos de la página XHTML cuyo atributo `name` coincida con el indicado como parámetro.

```
var elementos = document.getElementsByName("apellidos");
```

Una colección no es un array, aunque se le parezca mucho, ya que aunque puedas recorrerla y referenciar a sus elementos como un array, no se pueden usar métodos de array, como push o pop, en la colección.

Si sólo tenemos un elemento con `name="apellidos"` para acceder a él haremos: `var elemento = document.getElementsByName("apellidos")[0];`

Por ejemplo, si tuviéramos 3 elementos con el atributo `name="apellidos"` para acceder al segundo elemento haríamos: `var segundo = document.getElementsByName("apellidos")[1]; // recordarte que los arrays comienzan en la posición 0`.

Lo que nos permiten estas colecciones de elementos, es el poder recorrerlas fácilmente empleando un bucle, por ejemplo:

```
for (var j=1; j<document.getElementsByName("apellidos").length; j++)
{     var elemento = document.getElementsByName("apellidos")[j];  …..   }
```

getElementsByTagName()

Esta función es muy similar a la anterior y también devuelve una colección de elementos cuya etiqueta XHTML coincida con la que se pasa como parámetro. Por ejemplo:

```
var elementos = document.getElementsByTagName("input");
// Este array de elementos contendrá todos los elementos input del documento.

var cuarto = document.getElementsByTagName("input")[3];
```

getElementById()

Esta función es la más utilizada, ya que nos permite acceder directamente al elemento por el ID. Entre paréntesis escribiremos la cadena de texto con el ID. Es muy importante que el ID sea único para cada elemento de una misma página. La función nos devolverá únicamente el nodo buscado. Por ejemplo:

```
var elemento= document.getElementById("apellidos");
```

Si tenemos por ejemplo una tabla con `id="datos"` y queremos acceder a todas las celdas de esa tabla, tendríamos que combinar `getElementById` con `getElementsByTagName`. Por ejemplo:

```
var miTabla= document.getElementById("datos");
var filas= miTabla.getElementsByTagName("td");
```

1.4.- Acceso a los nodos de tipo atributo.

Una vez que ya hemos visto cómo acceder a los nodos (elementos XHTML) en un documento, vamos a ver cómo podemos acceder a los nodos de tipo atributo. Para referenciar un atributo, como por ejemplo el atributo `type="text"` del campo "apellidos", emplearemos la colección `attributes`. Dependiendo del navegador, esta colección se podrá cubrir de diferentes maneras y podrán existir muchos pares en la colección, tantos como atributos tenga el elemento. Para buscar el par correcto emplearemos la propiedad `nodeName`, que nos devolverá el nombre del atributo (en minúsculas cuando trabajamos con XHTML), y para acceder a su valor usaremos `nodeValue`.

En el ejemplo:

```
<input type="text" id="apellidos" name="apellidos" />
```

Para imprimir todos los atributos del elemento "apellidos", podríamos hacer un bucle que recorriera todos esos atributos imprimiendo su valor:

```
document.write("<br/>El elemento <b>apellidos</b> contiene los pares atributo -> valor:
<br/>");

for( var x = 0; x < document.getElementById("apellidos").attributes.length; x++ )
{
   var atributo = document.getElementById("apellidos").attributes[x];
   document.write(atributo.nodeName+ " -> "+atributo.nodeValue+"<br/>");
```

```
}
```

También podemos **modificar los valores de un atributo** de un nodo manualmente, por ejemplo:

```
document.getElementById("apellidos").attributes[0].nodeValue="password";
// En este caso hemos modificado el type del campo apellidos y lo hemos puesto de tipo "password".
```

O también:

```
document.getElementById("apellidos").attributes["type"].nodeValue="password";
// hemos puesto el nombre del atributo como referencia en el array de atributos.
```

O también:

```
document.getElementById("apellidos").type="password";
// hemos puesto el atributo como una propiedad del objeto apellidos y lo hemos modificado.
```

El método **setAttribute()** nos permitirá **crear o modificar atributos** de un elemento. Por ejemplo, para ponerle de nuevo al campo "apellidos" **type='text'** y un **value='Cid Blanco'**, haríamos:

```
document.getElementById("apellidos").setAttribute('type','text');
document.getElementById("apellidos").setAttribute('value','Cid Blanco');
```

Si lo que quieres realmente es chequear el valor del atributo y no modificarlo, se puede utilizar **getAttribute()**:

```
var valor = document.getElementById("apellidos").getAttribute('type');
// o también
var valor= document.getElementById("apellidos").type;
```

Y si lo que quieres es eliminar un atributo, lo podemos hacer con **removeAttribute()**:

```
// <div id="contenedor" align="left" width="200px">
document.getElementById("contenedor").removeAttribute("align");
// Obtendremos como resultado: <div id="contenedor" width="200px">
```

En XHTML los atributos se escribirán siempre en minúsculas.

Verdadero. ○ Falso. ○

Cuando escribimos los atributos en HTML, da igual que se pongan en mayúsculas o minúsculas, pero si estamos trabajando con XHTML deberemos escribirlos siempre en minúsculas (por ejemplo type, value, size, etc.).

1.5.- Acceso a los nodos de tipo texto.

Para ver cómo podemos acceder a la información textual de un nodo, nos basaremos en el siguiente ejemplo:

```
<p title="Texto de un párrafo">Esto es un ejemplo de <b>texto HTML<br />
que puedes tener</b> en tu documento.</p>
```

Para poder referenciar el fragmento `"texto HTML"` del nodo `P`, lo que haremos será utilizar la colección `childNodes`. Con la colección `childNodes` accederemos a los nodos hijo de un elemento, ya sean de tipo elemento o texto.

Aquí puedes ver una imagen del árbol DOM para ese elemento en cuestión:

Y el código de JavaScript para mostrar una alerta, con el contenido `"texto HTML"`, sería:

```
window.alert(document.getElementsByTagName("p")[0].childNodes[1].childNodes[0].nodeValue);
```

`childNodes[1]` : selecciona el segundo hijo de **`<p>`** que sería el elemento **`<b>`** (el primer hijo es un nodo de tipo Texto `"Esto es un..."`).
`childNodes[0]` : selecciona el primer hijo del elemento **`<b>`** que es el nodo de texto `"texto HTML"`

En lugar de **`childNodes[0]`** también podríamos haber utilizado **`firstChild`**, el cual nos devuelve el primer hijo de un nodo.

Por ejemplo:

```
window.alert(document.getElementsByTagName("p")[0].childNodes[1].firstChild.nodeValue);
```

El tamaño máximo de lo que se puede almacenar en un nodo de texto, depende del navegador, por lo que muchas veces, si el texto es muy largo, tendremos que consultar varios nodos para ver todo el contenido.

En el DOM de HTML, para acceder al valor de un nodo de texto, o modificarlo, es muy común ver la propiedad **`innerHTML`**. Esta propiedad es de Microsoft al igual que **`outerHTML`**. Aunque esta propiedad está soportada por casi todos los navegadores, y es muy rápida en su uso para hacer modificaciones, del contenido de un `DIV` por ejemplo, se recomienda usar el DOM si es posible.

Alternativas al uso de innerHTML. http://slayeroffice.com/articles/innerHTML_alternatives

Para modificar el contenido del nodo, modificaremos la propiedad **`nodeValue`** y le asignaremos otro valor. Por ejemplo en el caso anterior si hacemos:

```
document.getElementsByTagName("p")[0].childNodes[1].firstChild.nodeValue = "Texto MODIFICADO";
```

Veremos que en la página web se ha cambiado la cadena **`"texto HTML"`**, por **`"Texto MODIFICADO"`**.

También podríamos por ejemplo, mover trozos de texto a otras partes. El siguiente ejemplo mueve el texto `"en tu documento"` a continuación de `"Esto es un ejemplo de"`:

```
document.getElementsByTagName("p")[0].firstChild.nodeValue +=
document.getElementsByTagName("p")[0].childNodes[2].nodeValue;
document.getElementsByTagName("p")[0].childNodes[2].nodeValue="";
```

El resultado obtenido sería:

```
"Esto es un ejemplo de en tu documento.texto HTML
que puedes tener"
```

1.6.- Creación y borrado de nodos.

La creación y borrado de nodos fue uno de los objetivos para los que se creó el DOM. Podremos crear elementos y luego insertarlos en el DOM, y la actualización quedará reflejada automáticamente por el navegador. También podremos mover nodos ya existentes (como el párrafo del punto 1.4) simplemente insertándolo en cualquier otro lugar del árbol del DOM.

Ten en cuenta que cuando estemos creando nodos de elementos, el elemento debe estar en minúsculas. Aunque en HTML ésto daría igual, el XHTML sí que es sensible a mayúsculas y minúsculas y tendrá que ir, por lo tanto, en minúsculas.

Usaremos los métodos **`createElement()`**, **`createTextNode()`** y **`appendChild()`**, que nos permitirán crear un elemento, crear un nodo de texto y añadir un nuevo nodo hijo.

Ejemplo de creación de un nuevo párrafo, suponiendo que partimos del siguiente código HTML:

```
<p title="Texto de un párrafo" id="parrafito">Esto es un ejemplo de <b>texto HTML<br />
```

```
que puedes tener</b> en tu documento.</p>
```

Para crear el nuevo párrafo haremos:

```
var nuevoParrafo = document.createElement('p');
var nuevoTexto = document.createTextNode('Contenido añadido al párrafo.');
nuevoParrafo.appendChild(nuevoTexto);
document.getElementById('parrafito').appendChild(nuevoParrafo);
```

Y obtendremos como resultado HTML:

```
<p id="parrafito" title="Texto de un párrafo">
Esto es un ejemplo de <b>texto HTML<br>que puedes tener</b>en tu documento.
<p>Contenido añadido al párrafo.</p> </p>
```

Podríamos haber utilizado `insertBefore` en lugar de `appendChild` o, incluso, añadir manualmente el nuevo elemento al final de la colección de nodos `childNodes`. Si usamos `replaceChild`, incluso podríamos sobrescribir nodos ya existentes. También es posible copiar un nodo usando `cloneNode(true)`. Ésto devolverá una copia del nodo, pero no lo añade automáticamente a la colección `childNodes`.

Para eliminar un nodo existente, lo podremos hacer con `element.removeChild(referencia al nodo hijo)`.

Ejemplo de creación de elementos e inserción en el documento:

```
//Creamos tres elementos nuevos: p, b, br
var elementoP = document.createElement('p');
var elementoB = document.createElement('b');
var elementoBR = document.createElement('br');

//Le asignamos un nuevo atributo title al elementoP que hemos creado.
elementoP.setAttribute('title','Parrafo creado desde JavaScript');

//Preparamos los nodos de texto
var texto1 = document.createTextNode('Con JavaScript se ');
var texto2 = document.createTextNode('pueden realizar ');
var texto3 = document.createTextNode('un monton');
var texto4 = document.createTextNode(' de cosas sobre el documento.');

//Añadimos al elemento B los nodos de texto2, elemento BR y texto3.
elementoB.appendChild(texto2);
elementoB.appendChild(elementoBR);
elementoB.appendChild(texto3);

//Añadimos al elemento P los nodos de texto1, elemento B y texto 4.
elementoP.appendChild(texto1);
elementoP.appendChild(elementoB);
elementoP.appendChild(texto4);

//insertamos el nuevo párrafo como un nuevo hijo de nuestro parrafo
document.getElementById('parrafito').appendChild(elementoP);
```

1.7.- Propiedades y métodos de los objetos nodo (DOM nivel 2 W3C).

Propiedades:

Propiedad	Valor	Descripción	IE6Win+	IE5Mac+	Mozilla	Safari
nodeName	String	Varía según el tipo de nodo.	Sí.	Sí.	Sí.	Sí.
nodeValue	String	Varía según el tipo de nodo.	Sí.	Sí.	Sí.	Sí.
nodeType	Integer	Constante que representa cada tipo.	Sí.	Sí.	Sí.	Sí.
parentNode	Object	Referencia al siguiente contenedor más externo.	Sí.	Sí.	Sí.	Sí.
childNodes	Array	Todos los nodos hijos en orden.	Sí.	Sí.	Sí.	Sí.
firstChild	Object	Referencia al primer nodo	Sí.	Sí.	Sí.	Sí.

Propiedad	Valor	Descripción	IE6Win+	IE5Mac+	Mozilla	Safari
		hijo.				
lastChild	Object	Referencia al último nodo hijo.	Sí.	Sí.	Sí.	Sí.
previousSibling	Object	Referencia al hermano anterior según su orden en el código fuente.	Sí.	Sí.	Sí.	Sí.
nextSibling	Object	Referencia al hermano siguiente según su orden en el código fuente.	Sí.	Sí.	Sí.	Sí.
attributes	NodeMap	Array de atributos de los nodos.	Sí.	Algunos.	Sí.	Sí.
ownerDocument	Object	Contiene el objeto document.	Sí.	Sí.	Sí.	Sí.
namespaceURI	String	URI a la definición de namespace.	Sí.	No.	Sí.	Sí.
Prefix	String	Prefijo del namespace.	Sí.	No.	Sí.	Sí.
localName	String	Aplicable a los nodos afectados en el namespace.	Sí.	No.	Sí.	Sí.

Métodos:

Método	Descripción	IE5++	Mozilla	Safari
appendChild(newChild)	Añade un hijo al final del nodo actual.	Sí.	Sí.	Sí.
cloneNode(deep)	Realiza una copia del nodo actual (opcionalmente con todos sus hijos).	Sí.	Sí.	Sí.
hasChildNodes()	Determina si el nodo actual tiene o no hijos (valorboolean).	Sí.	Sí.	Sí.
insertBefore(new, ref)	Inserta un nuevo hijo antes de otro hijo.	Sí.	Sí.	Sí.
removeChild(old)	Borra un hijo.	Sí.	Sí.	Sí.
replaceChild(new, old)	Reemplaza un hijo viejo con el nuevo viejo.	Sí.	Sí.	Sí.
isSupported(feature, version)	Determina cuando el nodo soporta una característica especial.	No.	Sí.	Sí.

Ampliación de información de propiedades y métodos del objeto Nodo.
http://reference.sitepoint.com/javascript/Node