DOM II

La interfaz Node

Una vez que DOM ha creado de forma **automática** el **árbol** completo de **nodos** de la página, ya es posible utilizar sus funciones para **obtener información** sobre los nodos o **manipular su contenido**. JavaScript crea el **objeto Node** para definir las propiedades y métodos necesarios para procesar y manipular los documentos.

En primer lugar, el objeto Node define las siguientes **constantes** para la identificación de los **distintos tipos de nodos**:

```
Node.ELEMENT_NODE = 1
Node.ATTRIBUTE_NODE = 2
Node.TEXT_NODE = 3
Node.CDATA_SECTION_NODE = 4
Node.ENTITY_REFERENCE_NODE = 5
Node.ENTITY_NODE = 6
Node.PROCESSING_INSTRUCTION_NODE = 7
Node.COMMENT_NODE = 8
Node.DOCUMENT_NODE = 9
Node.DOCUMENT_TYPE_NODE = 10
Node.DOCUMENT_FRAGMENT_NODE = 11
Node.NOTATION_NODE = 12
```

Además de estas constantes, Node proporciona las siguientes propiedades y métodos:

Propiedad/Método	Valor devuelto	Descripción
nodeName	String	El nombre del nodo (no está definido para algunos tipos de nodo)
nodeValue	String	El valor del nodo (no está definido para algunos tipos de nodo)
nodeType	Number	Una de las 12 constantes definidas anteriormente

ownerDocument	Document	Referencia del documento al que pertenece el nodo
firstChild	Node	Referencia del primer nodo de la lista childNodes
lastChild	Node	Referencia del último nodo de la lista childNodes
childNodes	NodeList	Lista de todos los nodos hijo del nodo actual
previousSibling	Node	Referencia del nodo hermano anterior o null si este nodo es el primer hermano
nextSibling	Node	Referencia del nodo hermano siguiente o null si este nodo es el último hermano
hasChildNodes()	Bolean	Devuelve true si el nodo actual tiene uno o más nodos hijo
attributes	NamedNodeMap	Se emplea con nodos de tipo Element. Contiene objetos de tipo Attr que definen todos los atributos del elemento
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNodes
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes
replaceChild(nuevoNodo, anteriorNodo)	Node	Reemplaza el nodo anteriorNodo por el nodo nuevoNodo
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo nuevoNodo antes que la posición del nodo anteriorNodo dentro de la lista childNodes

Los métodos y propiedades incluidas en la tabla anterior son específicos de XML, aunque **pueden aplicarse a todos los lenguajes basados en XML**, como por ejemplo HTML. Para las páginas creadas con HTML, los **navegadores hacen como si HTML estuviera basado en XML** y lo tratan de la misma forma.

HTML y DOM

El uso de **DOM** siempre está **limitado** por las posibilidades que ofrece cada **navegador**. Hoy en día todos los navegadores implementan DOM de nivel 1 y 2 (y parte del 3), W3C publica las recomendaciones para un uso estándar de las distintas especificaciones de DOM.

Cuando se utiliza **DOM** en páginas **HTML**, el **nodo raíz** de todos los demás se define en el objeto **HTMLDocument**. Además, se crean objetos de tipo **HTMLElement** por **cada nodo de tipo Element** del árbol DOM.

El objeto **document** es parte del **BOM** (Browser Object Model), aunque también se considera que es **equivalente** del objeto **Document** del **DOM** de los documentos XML. Por este motivo, el objeto **document también hace referencia al nodo raíz de todas las páginas HTML**.

Acceso relativo a los nodos

A continuación se muestra la página HTML básica que se va a emplear en todos los siguientes ejemplos:

```
<html>
<head>
 <title>Aprendiendo DOM</title>
</head>
<body>
 <p>Aprendiendo DOM</p>
 <p>DOM es sencillo de aprender</p>
 <p>Ademas, DOM es muy potente</p>
</body>
</html>
```

La operación básica consiste en obtener el objeto que representa el **elemento raíz de la página**:

```
var objeto_html = document.documentElement;
```

Después de ejecutar la instrucción anterior, la variable **objeto**_html contiene un objeto de tipo **HTMLElement** y que representa el elemento <html> de la página web. Según el árbol de nodos DOM, desde el nodo <html> derivan dos nodos del mismo nivel jerárquico: <head> y <body>.

Utilizando los métodos proporcionados por DOM, es sencillo obtener los elementos <head> y <body>. En primer lugar, los dos nodos se pueden obtener como el **primer** y el **último nodo hijo** del elemento <html>:

```
var objeto_head = objeto_html.firstChild;
var objeto_body = objeto_html.lastChild;
```

Otra forma directa de obtener los dos nodos consiste en utilizar la **propiedad childNodes** del elemento <html>:

```
var objeto_head = objeto_html.childNodes[1];
var objeto_body = objeto_html.childNodes[2];
```

Si se desconoce el **número de nodos hijo** que dispone un nodo, se puede emplear la **propiedad length** de **childNodes**:

```
var numeroDescendientes = objeto_html.childNodes.length;
```

Además, el DOM de HTML permite **acceder directamente** al elemento <body> utilizando el atajo **document.body**:

```
var objeto_body = document.body;
```

Además de las propiedades anteriores, existen otras **propiedades** como **previousSibling** y **parentNode** que se pueden utilizar para acceder a un nodo a partir de otro. Utilizando estas propiedades, se pueden comprobar las siguientes igualdades:

```
objeto_head.parentNode == objeto_html
objeto_body.parentNode == objeto_html
objeto_body.previousSibling == objeto_head
objeto_head.nextSibling == objeto_body
objeto_head.ownerDocument == document
```

Tipos de nodos

Una operación común en muchas aplicaciones consiste en comprobar el **tipo de nodo**, que se obtiene de forma directa mediante la **propiedad nodeType**:

```
alert(document.nodeType); // 9
alert(document.documentElement.nodeType); // 1
```

En el primer caso, el valor 9 es igual al definido en la constante **Node.DOCUMENT_NODE**. En el segundo ejemplo, el valor 1 coincide con la constante **Node.ELEMENT_NODE**.

Afortunadamente, no es necesario memorizar los valores numéricos de los tipos de nodos, ya que se pueden emplear las constantes predefinidas:

```
alert(document.nodeType == Node.DOCUMENT_NODE); // true
alert(document.documentElement.nodeType == Node.ELEMENT_NODE); // true
```

Atributos

Además del tipo de etiqueta HTML y su contenido de texto, DOM permite el **acceso directo a todos los atributos de cada etiqueta**. Para ello, los nodos de tipo Element contienen la **propiedad attributes**, que permite acceder a todos los atributos de cada elemento. Aunque técnicamente la propiedad attributes es de **tipo NamedNodeMap**, sus elementos se pueden acceder **como si fuera un array**. DOM proporciona diversos métodos para tratar con los atributos:

- **getNamedItem(nombre)**, **devuelve** el **nodo** cuya **propiedad nodeName** contenga el valor **nombre**.
- **removeNamedItem(nombre)**, **elimina** el **nodo** cuya **propiedad nodeName** coincida con el valor **nombre**.
- **setNamedItem(nodo)**, **añade** el **nodo** a la lista **attributes**, **indexándolo según** su **propiedad nodeName**.
- **item(posicion)**, **devuelve** el **nodo** que se encuentra en la **posición** indicada por el valor numérico posicion.

Los métodos anteriores **devuelven un nodo de tipo Attr,** por lo que **no devuelven directamente el valor del atributo**. Utilizando estos métodos, es posible procesar y modificar fácilmente los atributos de los elementos HTML:

```
<p id="introduccion" style="color: blue">Párrafo de prueba</p>

var p = document.getElementById("introduccion");
var elId = p.attributes.getNamedItem("id").nodeValue; // elId =
"introduccion"
var elId = p.attributes.item(0).nodeValue; // elId =
"introduccion"
p.attributes.getNamedItem("id").nodeValue = "preintroduccion";

var atributo = document.createAttribute("lang");
atributo.nodeValue = "es";
p.attributes.setNamedItem(atributo);
```

Afortunadamente, DOM proporciona otros métodos que permiten el **acceso** y la **modificación** de los **atributos** de forma más **directa**:

- **getAttribute(nombre)**, equivale a **attributes.getNamedItem(nombre)**.
- **setAttribute(nombre, valor)** equivale a **attributes.getNamedItem(nombre).value =**

valor.

- **removeAttribute(nombre)**, equivale a **attributes.removeNamedItem(nombre)**. De

esta forma, el ejemplo anterior se puede reescribir utilizando los nuevos métodos: **<p**

```
id="introduccion" style="color: blue">Párrafo de prueba</p>


var p = document.getElementById("introduccion");
var elId = p.getAttribute("id"); // elId = "introduccion"
p.setAttribute("id", "preintroduccion");
```

Crear, modificar y eliminar nodos

Hasta ahora, todos los métodos DOM presentados se limitan a acceder a los nodos y sus propiedades. A continuación, se muestran los métodos necesarios para la **creación**, **modificación** y **eliminación** de **nodos** dentro del árbol de nodos DOM.

Los **métodos más empleados** son **createElement**, **createTextNode** y **createAttribute**.

Una vez más, es importante recordar que las **modificaciones** en el árbol de nodos DOM sólo se pueden realizar **cuando toda la página web se ha cargado en el navegador**. El motivo es que los navegadores construyen el árbol de nodos DOM una vez que se ha cargado completamente la página web. Cuando una página no ha terminado de cargarse, su árbol no está construido y por tanto no se pueden utilizar las funciones DOM.

Si una página realiza **modificaciones automáticas** (sin intervención del usuario) es importante utilizar el **evento onload()** para llamar a las funciones de JavaScript, tal y como se verá más adelante en el capítulo de los eventos.

Además de crear y eliminar nodos como vimos en el documento anterior, las funciones DOM también permiten **reemplazar un nodo por otro**. En la página HTML de ejemplo, se va a sustituir el párrafo original por otro párrafo con un contenido diferente. La página original contiene el siguiente párrafo:

```
<html>
 <head><title>Ejemplo de sustitución de nodos</title></head>
<body>
 <p>Este parrafo va a ser sustituido dinamicamente</p>
</body>
</html>
```

Utilizando las funciones DOM de JavaScript, se crea el nuevo párrafo que se va a mostrar en la página, se obtiene la referencia al nodo original y se emplea la **función replaceChild()** para intercambiar un nodo por otro:

```
var nuevoP = document.createElement("p");
var texto = document.createTextNode("Este parrafo se ha creado
dinámicamente y sustituye al parrafo original");
nuevoP.appendChild(texto);

var anteriorP =
document.body.getElementsByTagName("p")[0];
anteriorP.parentNode.replaceChild(nuevoP, anteriorP);
```

Después de ejecutar las instrucciones anteriores, la página HTML resultante es:

```
<html>
 <head><title>Ejemplo de sustitución de nodos</title></head>
<body>
 <p>Este parrafo se ha creado dinámicamente y sustituye al parrafo
original</p>
 </body>
</html>
```

Si se quiere insetar un **nuevo párrafo delante** del párrafo existente, se puede utilizar la **función insertBefore()**. Página HTML original:

```
<html>
 <head><title>Ejemplo de inserción de nodos</title></head>
<body>
 <p>Primer párrafo</p>
 </body>
</html>
```

Código JavaScript necesario para insertar un **nuevo párrafo delante** del párrafo existente:

```
var nuevoP = document.createElement("p");
var texto = document.createTextNode("Segundo párrafo, antes del
primero"); nuevoP.appendChild(texto);

var anteriorP = document.getElementsByTagName("p")[0];
document.body.insertBefore(nuevoP, anteriorP);
```

Después de ejecutar el código JavaScript anterior, el código HTML resultante es:

```
<html>
 <head><title>Ejemplo de inserción de nodos</title></head>
<body>
 <p>Segundo párrafo, antes del primero</p>
 <p>Primer párrafo</p>
 </body>
</html>
```

Atributos HTML y propiedades CSS en DOM

Los métodos presentados anteriormente para el acceso a los atributos de los elementos, son genéricos de XML. La versión de DOM específica para HTML incluye algunas propiedades y **métodos** aún más **directos** y **sencillos** para el acceso a los atributos de los **elementos HTML** y a sus **propiedades CSS**.

La principal ventaja del DOM para HTML es que todos los atributos de todos los **elementos HTML se transforman en propiedades de los nodos**. De esta forma, es posible acceder de forma directa a cualquier atributo de HTML. Si se considera el siguiente elemento <img> de HTML con sus tres atributos:

```
<img id="logo" src="logo.gif" border="0" />
```

Empleando los métodos tradicionales de DOM, se puede acceder y manipular cada atributo:

```
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.getAttribute("src");
var borde = laImagen.getAttribute("border");

// modificar los atributos
laImagen.setAttribute("src", "nuevo_logo.gif");
laImagen.setAttribute("border", "1");
```

La ventaja de la especificación de DOM para HTML es que permite **acceder** y **modificar** todos los **atributos** de los elementos de **forma directa**:

```
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.src;
var borde = laImagen.border;

// modificar los atributos
laImagen.src = "nuevo_logo.gif";
laImagen.border = "1";
```

Las ventajas de utilizar esta forma de acceder y modificar los atributos de los elementos es que el **código resultante** es **más sencillo** y **conciso**. Por otra parte, algunas versiones de Internet Explorer no implementan correctamente el método **setAttribute()**, lo que provoca que, en ocasiones, los cambios realizados no se reflejan en la página HTML.

La única **excepción** que existe en esta forma de obtener el valor de los atributos HTML es el **atributo class**. Como la palabra class está reservada por JavaScript para su uso futuro, no es posible utilizarla para acceder al valor del atributo class de HTML. La solución consiste en acceder a ese atributo mediante el **nombre alternativo className**:

```
<p id="parrafo" class="normal">...</p>

var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"
```

El **acceso** a las **propiedades CSS** no es tan directo y sencillo como el acceso a los atributos HTML. En primer lugar, los estilos CSS se pueden aplicar de varias formas diferentes sobre un mismo elemento HTML. Si se establecen las propiedades CSS mediante el atributo style de HTML:

```
<p id="parrafo" style="color: #C00">...</p>
```

El acceso a las propiedades CSS establecidas mediante el atributo style se realiza a través de la **propiedad style** del nodo que representa a ese elemento:

```
var parrafo = document.getElementById("parrafo");
var color = parrafo.style.color;
```

Para acceder al valor de una propiedad CSS, se **obtiene** la **referencia** del **nodo**, se **accede** a su **propiedad style** y a continuación se **indica** el **nombre** de la **propiedad CSS** cuyo valor se quiere obtener. Aunque parece lógico que en el ejemplo anterior el valor obtenido sea #C00, en realidad **cada navegador obtiene** el mismo **valor** de **formas diferentes**:

- Firefox y Safari muestran el valor rgb(204, 0, 0)
- Internet Explorer muestra el valor #c00
- Opera muestra el valor #cc0000