4.4. HTML y DOM

Desafortunadamente, las posibilidades teóricas de DOM son mucho más avanzadas de las que se pueden utilizar en la práctica para desarrollar aplicaciones web. El motivo es que el uso de DOM siempre está limitado por las posibilidades que ofrece cada navegador. Mientras que algunos navegadores como Firefox y Safari implementan DOM de nivel 1 y 2 (y parte del 3), otros navegadores como Internet Explorer (versión 7 y anteriores) ni siquiera son capaces de ofrecer una implementación completa de DOM nivel 1.

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. Como se verá en el siguiente capítulo, 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.

4.4.1. 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[0];

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

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
```

4.4.2. 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
```

El único navegador que no soporta las constantes predefinidas es Internet Explorer 7 y sus versiones anteriores, por lo que si se quieren utilizar es necesario definirlas de forma explícita:

```
if(typeof Node == "undefined") {

  var Node = {

    ELEMENT_NODE: 1,

    ATTRIBUTE_NODE: 2,

    TEXT_NODE: 3,

    CDATA_SECTION_NODE: 4,

    ENTITY_REFERENCE_NODE: 5,

    ENTITY_NODE: 6,

    PROCESSING_INSTRUCTION_NODE: 7,

    COMMENT_NODE: 8,

    DOCUMENT_NODE: 9,

    DOCUMENT_TYPE_NODE: 10,

    DOCUMENT_FRAGMENT_NODE: 11,

    NOTATION_NODE: 12

  };}
```

El código anterior comprueba si el navegador en el que se está ejecutando tiene definido el objeto `Node`. Si este objeto no está definido, se trata del navegador Internet Explorer 7 o alguna versión anterior, por lo que se crea un nuevo objeto llamado `Node` y se le incluyen como propiedades todas las constantes definidas por DOM.

4.4.3. 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)`, es equivalente a `attributes.getNamedItem(nombre)`.
- `setAttribute(nombre, valor)` equivalente a `attributes.getNamedItem(nombre).value = valor`.
- `removeAttribute(nombre)`, equivalente 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");
```

4.4.4. Acceso directo a los nodos

Los métodos presentados hasta el momento permiten acceder a cualquier nodo del árbol de nodos DOM y a todos sus atributos. Sin embargo, las funciones que proporciona DOM para acceder a un nodo a través de sus padres obligan a acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado.

Cuando se trabaja con una página web real, el árbol DOM tiene miles de nodos de todos los tipos. Por este motivo, no es eficiente acceder a un nodo descendiendo a través de todos los ascendentes de ese nodo.

Para solucionar este problema, DOM proporciona una serie de métodos para acceder de forma directa a los nodos deseados. Los métodos disponibles son `getElementsByTagName()`, `getElementsByName()` y `getElementById()`.

4.4.4.1. Función getElementsByTagName()

La función `getElementsByTagName()` obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. En realidad, el valor devuelto no es de tipo array normal, sino que es un objeto de tipo `NodeList`. De este modo, el primer párrafo de la página se puede obtener de la siguiente manera:

```
var parrafos = document.getElementsByTagName("p");

var primerParrafo = parrafos[0];
```

De la misma forma, se pueden recorrer todos los párrafos de la página recorriendo el array de nodos devuelto por la función:

```
var parrafos = document.getElementsByTagName("p");

for(var i=0; i<parrafos.length; i++) {

  var parrafo = parrafos[i];

}
```

La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");

var primerParrafo = parrafos[0];

var enlaces = primerParrafo.getElementsByTagName("a");
```

4.4.4.2. Función getElementsByName()

La función `getElementsByName()` obtiene todos los elementos de la página XHTML cuyo atributo `name` coincida con el parámetro que se le pasa a la función.

En el siguiente ejemplo, se obtiene directamente el único párrafo de la página que tiene el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");
```

```
<p name="prueba">...</p>

<p name="especial">...</p>

<p>...</p>
```

Normalmente el atributo `name` es único para los elementos HTML que lo incluyen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML `radiobutton`, el atributo `name` es común a todos los `radiobutton` que están relacionados, por lo que la función devuelve una colección de elementos.

Internet Explorer 7 y sus versiones anteriores no implementan de forma correcta esta función, ya que también devuelven los elementos cuyo atributo `id` sea igual al parámetro de la función.

4.4.4.3. Función getElementById()

La función `getElementById()` es la función más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y para leer o modificar sus propiedades.

La función `getElementById()` devuelve el elemento XHTML cuyo atributo `id` coincide con el parámetro indicado en la función. Como el atributo `id` debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");
```

```
<div id="cabecera">
<a href="/" id="logo">...</a>
</div>
```

Ejercicio 7

A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:

1. Número de enlaces de la página
2. Dirección a la que enlaza el penúltimo enlace
3. Numero de enlaces que enlazan a http://prueba
4. Número de enlaces del tercer párrafo

Descargar archivo ZIP con la página HTML (2 KB)

Ver solución

Ejercicio 8

A partir de la página web proporcionada y utilizando las funciones DOM:

1. Se debe modificar el protocolo de todas las direcciones de los enlaces. De esta forma, si un enlace apuntaba a http://prueba, ahora debe apuntar a https://prueba
2. Los párrafos de la página cuyo atributo `class` es igual a `"importante"` deben modificarlo por `"resaltado"`. El resto de párrafos deben incluir un atributo `class` igual a `"normal"`.
3. A los enlaces de la página cuyo atributo `class` sea igual a `"importante"`, se les añade un atributo `"name"` con un valor generado automáticamente y que sea igual a `"importante"+i`, donde `i` es un valor numérico cuyo valor inicial es `0` para el primer enlace.

Descargar archivo ZIP con la página HTML (2 KB)

Ver solución

4.4.5. 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 DOM disponibles para la creación de nuevos nodos son los siguientes:

Método	Descripción
`createAttribute(nombre)`	Crea un nodo de tipo atributo con el nombre indicado
`createCDataSection(texto)`	Crea una sección `CDATA` con un nodo hijo de tipo texto que contiene el valor indicado
`createComment(texto)`	Crea un nodo de tipo comentario que contiene el valor indicado
`createDocumentFragment()`	Crea un nodo de tipo `DocumentFragment`
`createElement(nombre_etiqueta)`	Crea un elemento del tipo indicado en el parámetro `nombre_etiqueta`
`createEntityReference(nombre)`	Crea un nodo de tipo `EntityReference`
`createProcessingInstruction(objetivo, datos)`	Crea un nodo de tipo `ProcessingInstruction`
`createTextNode(texto)`	Crea un nodo de tipo texto con el valor indicado como parámetro

Internet Explorer no soporta los métodos `createCDataSection`, `createEntityReference` y `createProcessingInstruction`. Los métodos más empleados son `createElement`, `createTextNode` y `createAttribute`.

A continuación se muestra un ejemplo sencillo en el que se crea un nuevo nodo (una nueva etiqueta HTML) y se añade dinámicamente a la siguiente página HTML:

```
<html>

  <head><title>Ejemplo de creación de nodos</title></head>

  <body></body>

</html>
```

La página HTML original no tiene contenidos, pero mediante DOM se añade dinámicamente el siguiente párrafo de texto:

```
<p>Este párrafo no existía en la página HTML original</p>
```

Añadir el párrafo anterior a la página requiere los siguientes pasos:

1. Crear un nodo de tipo elemento
2. Crear un nodo de tipo texto
3. Asociar el nodo de texto al elemento
4. Añadir el nuevo nodo de tipo elemento a la página original

En primer lugar, se crea un nuevo nodo de tipo elemento:

```
var p = document.createElement("p");
```

A continuación se crea un nodo de texto que almacena el contenido de texto del párrafo:

```
var texto = document.createTextNode("Este párrafo no existía en la página HTML

original");
```

En tercer lugar, se asocia el elemento creado y su contenido de texto (los nodos de tipo `Text` son hijos de los nodos de tipo `Element`):

```
p.appendChild(texto);
```

El método `appendChild()` está definido para todos los diferentes tipos de nodos y se encarga de añadir un nodo al final de la lista `childNodes` de otro nodo.

Por último, se añade el nodo creado al árbol de nodos DOM que representa a la página. Utilizando el método `appendChild()`, se añade el nodo como hijo del nodo que representa al elemento `<body>` de la página:

```
document.body.appendChild(p);
```

La página HTML que resulta después de la ejecución del código JavaScript se muestra a continuación:

```
<html>
  <head><title>Ejemplo de creación de nodos</title></head>
  <body>
    <p>Este párrafo no existía en la página HTML original</p>
  </body>
</html>
```

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.

Por otra parte, también se pueden utilizar funciones DOM para eliminar cualquier nodo existente originalmente en la página y cualquier nodo creado mediante los métodos DOM:

Si se considera la siguiente página HTML con un párrafo de texto:

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

En primer lugar, se obtiene la referencia al nodo que se va a eliminar:

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

Para eliminar cualquier nodo, se emplea la función `removeChild()`, que toma como argumento la referencia al nodo que se quiere eliminar. La función `removeChild()` se debe invocar sobre el nodo padre del nodo que se va a eliminar. En este caso, el padre del primer párrafo de la página es el nodo `<body>`:

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

```
document.body.removeChild(p);
```

La página HTML que resulta después de la ejecución del código JavaScript anterior se muestra a continuación:

```
<html>

  <head><title>Ejemplo de eliminación de nodos</title></head>

  <body></body>

</html>
```

Cuando la página está formada por miles de nodos, puede ser costoso acceder hasta el nodo padre del nodo que se quiere eliminar. En estos casos, se puede utilizar la propiedad `parentNode`, que siempre hace referencia al nodo padre de un nodo.

De esta forma, el ejemplo anterior se puede rehacer para eliminar el nodo haciendo uso de la propiedad `parentNode`:

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

p.parentNode.removeChild(p);
```

Además de crear y eliminar nodos, las funciones DOM también permiten reemplazar un nodo por otro. Utilizando la misma 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>
```

Además de crear, eliminar y sustituir nodos, las funciones DOM permiten insertar nuevos nodos antes o después de otros nodos ya existentes. Si se quiere insertar un nodo después de otro, se emplea la función `appendChild()`. Página HTML original:

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

El siguiente código JavaScript crea un nuevo párrafo y lo inserta después de cualquier otro nodo de la página HTML:

```
var nuevoP = document.createElement("p");

var texto = document.createTextNode("Segundo parrafo");

nuevoP.appendChild(texto);


document.body.appendChild(nuevoP);
```

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

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

Si se quiere insetar el 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 parrafo</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 parrafo, 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 parrafo, antes del primero</p>
    <p>Primer parrafo</p>
```

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

Ejercicio 9

A partir de la página HTML que se proporciona, completar el código JavaScript definido para realizar la siguiente aplicación sencilla:

Figura 4.4 Inicio de la aplicación

Figura 4.5 Generación de números aleatorios

Figura 4.6 Resultado de la comparación

Inicialmente, la aplicación cuenta con tres cajas vacías y dos botones. Al presionar el botón de "Genera", se crean dos números aleatorios. Cada número aleatorio se guarda en un elemento `<p>`, que a su vez se guarda en una de las dos cajas superiores.

Una vez generados los números, se presiona el botón "Comparar", que compara el valor de los dos párrafos anteriores y determina cual es el mayor. El párrafo con el número más grande, se mueve a la última caja que se utiliza para almacenar el resultado de la operación.

La página que se proporciona contiene todo el código HTML y CSS necesario. Además, incluye todo el código JavaScript relativo a la pulsación de los botones y que se estudiará con detalle en el siguiente capítulo.

En el ejercicio se deben utilizar entre otras, las funciones `getElementById()`, `createElement()`, `createTextNode()`, `appendChild()` y `replaceChild()`.

Descargar archivo ZIP con la página HTML

Ver solución

4.4.6. 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`

En el caso de las propiedades CSS con nombre compuesto (`font-weight`, `border-top-style`, `list-style-type`, etc.), para acceder a su valor es necesario modificar su nombre original eliminando los guiones medios y escribiendo en mayúsculas la primera letra de cada palabra que no sea la primera:

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

var negrita = parrafo.style.fontWeight;
```

El nombre original de la propiedad CSS es `font-weight`. Para obtener su valor mediante JavaScript, se elimina el guión medio (`fontweight`) y se pasa a mayúsculas la primera letra de cada palabra que no sea la primera (`fontWeight`).

Si el nombre de la propiedad CSS está formado por tres palabras, se realiza la misma transformación. De esta forma, la propiedad `border-top-style` se accede en DOM mediante el nombre `borderTopStyle`.

Además de obtener el valor de las propiedades CSS, también es posible modificar su valor mediante JavaScript. Una vez obtenida la referencia del nodo, se puede modificar el valor de cualquier propiedad CSS accediendo a ella mediante la propiedad `style`:

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


var parrafo = document.getElementById("parrafo");

parrafo.style.margin = "10px";   // añade un margen de 10px al párrafo

parrafo.style.color = "#CCC";    // modifica el color de la letra del párrafo
```

Todos los ejemplos anteriores hacen uso de la propiedad `style` para acceder o establecer el valor de las propiedades CSS de los elementos. Sin embargo, esta propiedad sólo permite acceder al valor de las propiedades CSS establecidas directamente sobre el elemento HTML. En otras palabras, la propiedad `style` del nodo sólo contiene el valor de las propiedades CSS establecidas mediante el atributo `style` de HTML.

Por otra parte, los estilos CSS normalmente se aplican mediante reglas CSS incluidas en archivos externos. Si se utiliza la propiedad `style` de DOM para acceder al valor de una propiedad CSS establecida mediante una regla externa, el navegador no obtiene el valor correcto:

```
// Código HTML

<p id="parrafo">...</p>


// Regla CSS
```

```
#parrafo { color: #008000; }


// Código JavaScript

var parrafo = document.getElementById("parrafo");

var color = parrafo.style.color;  // color no almacena ningún valor
```

Para obtener el valor de las propiedades CSS independientemente de cómo se hayan aplicado, es necesario utilizar otras propiedades de JavaScript. Si se utiliza un navegador de la familia Internet Explorer, se hace uso de la propiedad `currentStyle`. Si se utiliza cualquier otro navegador, se puede emplear la función `getComputedStyle()`.

```
// Código HTML

<p id="parrafo">...</p>


// Regla CSS

#parrafo { color: #008000; }


// Código JavaScript para Internet Explorer

var parrafo = document.getElementById("parrafo");

var color = parrafo.currentStyle['color'];


// Código JavaScript para otros navegadores

var parrafo = document.getElementById("parrafo");

var color = document.defaultView.getComputedStyle(parrafo,

'').getPropertyValue('color');
```

La propiedad `currentStyle` requiere el nombre de las propiedades CSS según el formato de JavaScript (sin guiones medios), mientras que la función `getPropertyValue()` exige el uso del nombre original de la propiedad CSS. Por este motivo, la creación de aplicaciones compatibles con todos los navegadores se puede complicar en exceso.

A continuación se muestra una función compatible con todos los navegadores, creada por el programador Robert Nyman y publicada en su blog personal:

```
function getStyle(elemento, propiedadCss) {

  var valor = "";
```

```
  if(document.defaultView && document.defaultView.getComputedStyle){
    valor = document.defaultView.getComputedStyle(elemento,
'').getPropertyValue(propiedadCss);
  }
  else if(elemento.currentStyle) {
    propiedadCss = propiedadCss.replace(/\-(\w)/g, function (strMatch, p1) {
      return p1.toUpperCase();
    });
    valor = elemento.currentStyle[propiedadCss];
  }
  return valor;
}
```

Utilizando la función anterior, es posible rehacer el ejemplo para que funcione correctamente en cualquier navegador:

```
// Código HTML
<p id="parrafo">...</p>

// Regla CSS
#parrafo { color: #008000; }

// Código JavaScript para cualquier navegador
var parrafo = document.getElementById("parrafo");
var color = getStyle(parrafo, 'color');
```

4.4.7. Tablas HTML en DOM

Las tablas son elementos muy comunes en las páginas HTML, por lo que DOM proporciona métodos específicos para trabajar con ellas. Si se utilizan los métodos tradicionales, crear una tabla es una tarea tediosa, por la gran cantidad de nodos de tipo elemento y de tipo texto que se deben crear.

Afortunadamente, la versión de DOM para HTML incluye varias propiedades y métodos para crear tablas, filas y columnas de forma sencilla.

Propiedades y métodos de `<table>`:

Propiedad/Método	Descripción
`rows`	Devuelve un array con las filas de la tabla
`tBodies`	Devuelve un array con todos los `<tbody>` de la tabla
`insertRow(posicion)`	Inserta una nueva fila en la posición indicada dentro del array de filas de la tabla
`deleteRow(posicion)`	Elimina la fila de la posición indicada

Propiedades y métodos de `<tbody>`:

Propiedad/Método	Descripción
`rows`	Devuelve un array con las filas del `<tbody>` seleccionado
`insertRow(posicion)`	Inserta una nueva fila en la posición indicada dentro del array de filas del `<tbody>`
`deleteRow(posicion)`	Elimina la fila de la posición indicada

Propiedades y métodos de `<tr>`:

Propiedad/Método	Descripción
`cells`	Devuelve un array con las columnas de la fila seleccionada
`insertCell(posicion)`	Inserta una nueva columna en la posición indicada dentro del array de columnas de la fila
`deleteCell(posicion)`	Elimina la columna de la posición indicada

Si se considera la siguiente tabla XHTML:

```
<table summary="Descripción de la tabla y su contenido">

<caption>Título de la tabla</caption>

<thead>

  <tr>

    <th scope="col"></th>

    <th scope="col">Cabecera columna 1</th>

    <th scope="col">Cabecera columna 2</th>

  </tr>

</thead>
```

```
<tfoot>
  <tr>
    <th scope="col"></th>
    <th scope="col">Cabecera columna 1</th>
    <th scope="col">Cabecera columna 2</th>
  </tr>
</tfoot>

<tbody>
  <tr>
    <th scope="row">Cabecera fila 1</th>
    <td>Celda 1 - 1</td>
    <td>Celda 1 - 2</td>
  </tr>
  <tr>
    <th scope="row">Cabecera fila 2</th>
    <td>Celda 2 - 1</td>
    <td>Celda 2 - 2</td>
  </tr>
</tbody>

</table>
```

A continuación se muestran algunos ejemplos de manipulación de tablas XHTML mediante las propiedades y funciones específicas de DOM.

Obtener el número total de filas de la tabla:

```
var tabla = document.getElementById('miTabla');
var numFilas = tabla.rows.length;
```

Obtener el número total de cuerpos de la tabla (secciones `<tbody>`):

```
var tabla = document.getElementById('miTabla');
var numCuerpos = tabla.tBodies.length;
```

Obtener el número de filas del primer cuerpo (sección `<tbody>`) de la tabla:

```
var tabla = document.getElementById('miTabla');
var numFilasCuerpo = tabla.tBodies[0].rows.length;
```

Borrar la primera fila de la tabla y la primera fila del cuerpo (sección `<tbody>`):

```
var tabla = document.getElementById('miTabla');
tabla.deleteRow(0);
tabla.tBodies[0].deleteRow(0);
```

Borrar la primera columna de la primera fila del cuerpo (sección `<tbody>`):

```
var tabla = document.getElementById('miTabla');
tabla.tBodies[0].rows[0].deleteCell(0);
```

Obtener el número de columnas de la primera parte del cuerpo (sección `<tbody>`):

```
var tabla = document.getElementById('miTabla');
var numColumnas = tabla.rows[0].cells.length;
```

Obtener el texto de la primera columna de la primera fila del cuerpo (sección `<tbody>`):

```
var tabla = document.getElementById('miTabla');
var texto = tabla.tBodies[0].rows[0].cells[0].innerHTML;
```

Recorrer todas las filas y todas las columnas de la tabla:

```
var tabla = document.getElementById('miTabla');
var filas = tabla.rows;
for(var i=0; i<filas.length; i++) {
  var fila = filas[i];
  var columnas = fila.cells;
```

```
  for(var j=0; j<columnas.length; j++) {
    var columna = columnas[j];
    // ...
  }
}
```

Insertar una tercera fila al cuerpo (sección `<tbody>`) de la tabla:

```
var tabla = document.getElementById('miTabla');

// Insertar la tercera fila
tabla.tBodies[0].insertRow(2);

// Crear la columna de tipo <th> que hace de cabecera de fila
var cabecera = document.createElement("th");
cabecera.setAttribute('scope', 'row');
cabecera.innerHTML = 'Cabecera fila 3'
tabla.tBodies[0].rows[2].appendChild(cabecera);

// Crear las dos columnas de datos y añadirlas a la nueva fila
tabla.tBodies[0].rows[2].insertCell(1);
tabla.tBodies[0].rows[2].cells[1].innerHTML = 'Celda 3 - 1';
tabla.tBodies[0].rows[2].insertCell(2);
tabla.tBodies[0].rows[2].cells[2].innerHTML = 'Celda 3 - 2';
```

Por último, se muestra de forma resumida el código JavaScript necesario para crear la tabla XHTML del ejemplo anterior:

```
// Crear <table> y sus dos atributos
var tabla = document.createElement('table');
tabla.setAttribute('id', 'otraTabla');
tabla.setAttribute('summary', 'Descripción de la tabla y su contenido');
```

```
// Crear <caption> y añadirlo a la <table>
var caption = document.createElement('caption');
var titulo = document.createTextNode('Título de la tabla');
caption.appendChild(titulo);
tabla.appendChild(caption);

// Crear sección <thead>
var thead = document.createElement('thead');
tabla.appendChild(thead);

// Añadir una fila a la sección <thead>
thead.insertRow(0);

// Añadir las tres columnas de la fila de <thead>
var cabecera = document.createElement('th');
cabecera.innerHTML = '';
thead.rows[0].appendChild(cabecera);

cabecera = document.createElement('th');
cabecera.setAttribute('scope', 'col');
cabecera.innerHTML = 'Cabecera columna 1';
tabla.rows[0].appendChild(cabecera);

cabecera = document.createElement('th');
cabecera.setAttribute('scope', 'col');
cabecera.innerHTML = 'Cabecera columna 2';
tabla.rows[0].appendChild(cabecera);

// La sección <tfoot> se crearía de forma similar a <thead>
```

```
// Crear sección <tbody>
var tbody = document.createElement('tbody');
tabla.appendChild(tbody);

// Añadir una fila a la sección <tbody>
tbody.insertRow(0);

cabecera = document.createElement("th");
cabecera.setAttribute('scope', 'row');
cabecera.innerHTML = 'Cabecera fila 1'
tabla.tBodies[0].rows[0].appendChild(cabecera);

tbody.rows[0].insertCell(1);
tbody.rows[0].cells[1].innerHTML = 'Celda 1 - 1';
// También se podría hacer:
// tbody.rows[0].cells[0].appendChild(document.createTextNode('Celda 1 - 1'));

tbody.rows[0].insertCell(2);
tbody.rows[0].cells[2].innerHTML = 'Celda 1 - 2';

// El resto de filas del <tbody> se crearía de la misma forma

// Añadir la tabla creada al final de la página
document.body.appendChild(tabla);
```