DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 6:

Utilización del modelos de objetos del documento (DOM-Document Object Model)

Juan Manuel Vara Mesa
Marcos López Sanz
David Granada
Emanuel Irrazábal
Jesús Javier Jiménez Hernández
Jenifer Verde Marín

Ra-Ma®

El modelo de objetos del documento (DOM)

- Es un estándar de W3C que define cómo acceder a los documentos, como por ejemplo HTML y XML.
- Es una interfaz de programación de aplicaciones (API) de la plataforma de W3C.
- Permite a los scripts acceder y actualizar dinámicamente su contenido, estructura y estilo de documento.

El modelo de objetos del documento (DOM)

- Fue utilizado por primera vez con el navegador Netscape Navigator V.2.0.
- A esta primera versión de DOM se le denomina DOM nivel 0.
- El primer navegador de Microsoft que utilizó el DOM nivel 0 fue IE 3.0.

El modelo de objetos del documento (DOM)

- Debido a las diferencias entre los navegadores, W3C emitió una especificación a finales de 1998 que llamó DOM nivel 1.
- En esta especificación ya se consideraba la manipulación de todos los elementos existentes en los archivos HTML.

El modelo de objetos del documento (DOM)

- A finales del año 2000, W3C emitió DOM nivel 2, en la cual se incluía el manejo de eventos en el navegador y la interacción con hojas de estilo CSS.
- En 2004 se emitió DOM nivel 3, en la cual se utiliza la definición de tipos de documento (DTD) y la validación de documentos.

El modelo de objetos del documento (DOM)

- Actualmente DOM se divide en tres partes según la W3C:
  - Núcleo del DOM.
  - XML DOM.
  - HTML DOM.

El modelo de objetos del documento (DOM)

- Núcleo del DOM:
  - Este es el modelo estándar para cualquier documento estructurado.
  - En este modelo se especifican a nivel general las pautas para definir los objetos y propiedades de cualquier documento estructurado así como los métodos para acceder a ellos.

El modelo de objetos del documento (DOM)

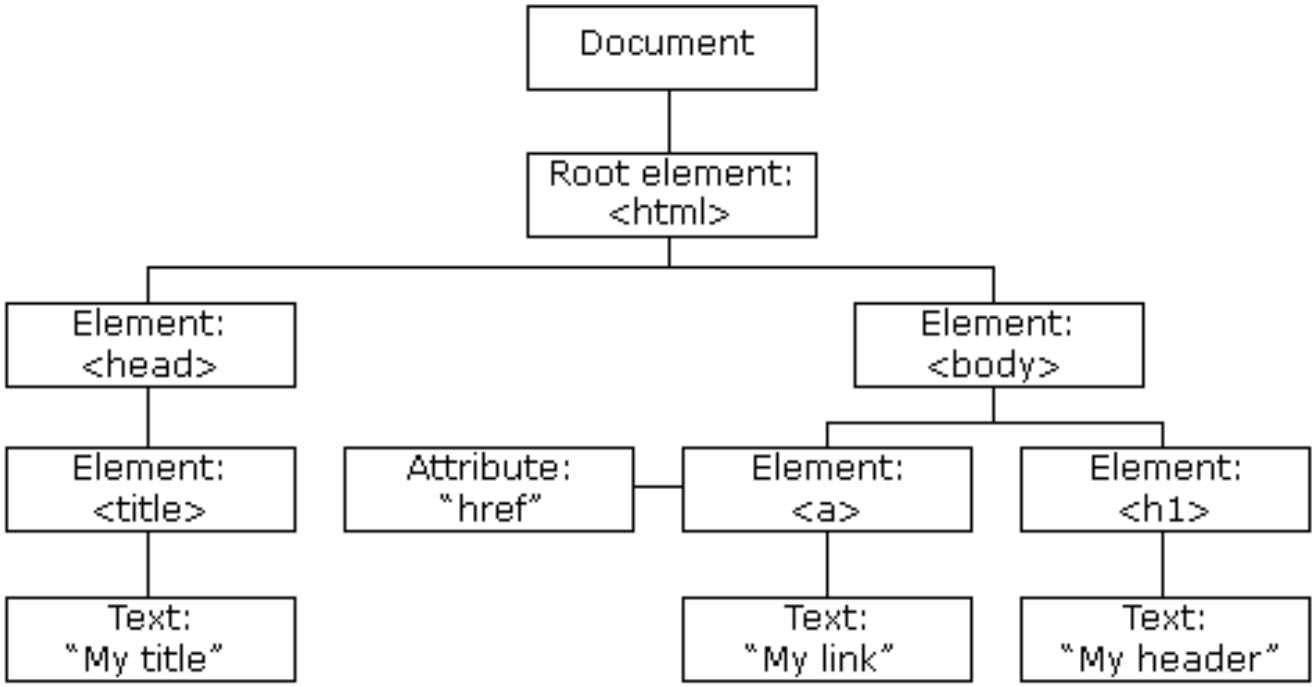
- XML DOM:
  - Este es el modelo estándar para los documentos XML.
  - Este modelo define los objetos y propiedades de todos los elementos XML, así como los métodos para acceder a ellos.

El modelo de objetos del documento (DOM)

- HTML DOM:
  - Un modelo de datos estándar para HTML.
  - Una interfaz de programación estándar para HTML.
  - Independiente de la plataforma y el lenguaje.
  - Es un estándar de la W3C.
  - El DOM HTML define los objetos y las propiedades de los elementos HTML además de definir los métodos para acceder a ellos.
  - Es un estándar sobre la forma de obtener, modificar, añadir o eliminar elementos HTML.

El modelo de objetos del documento (DOM)

- Estructura del árbol DOM:



El modelo de objetos del documento (DOM)

- Para ordenar la estructura del árbol, existe una serie de reglas:
  - En el árbol de nodos, al nodo superior (document), se le llama raíz.
  - Cada nodo, exceptuando el nodo raíz, tiene un padre.
  - Un nodo puede tener cualquier número de hijos.
  - Una hoja es un nodo sin hijos.
  - Los nodos que comparten el mismo padre, son hermanos.

Objetos del modelo, propiedades y métodos de los objetos

- Objetos del modelo (1):
  - **Document**. Es el nodo raíz del documento HTML. Todos los elementos del árbol cuelgan de él.
  - **DocumentType**. Este nodo indica la representación del DTD de la página. Un DTD es una definición de tipo de documento. Define la estructura y sintaxis de un documento XML. El *DOCTYPE* es el encargado de indicar el DocumentType.

Objetos del modelo, propiedades y métodos de los objetos

- Objetos del modelo (2):
  - **Element**. Este nodo representa el contenido de una pareja de etiquetas de apertura y cierre (`<etiqueta>…</etiqueta>`). También puede representar una etiqueta abreviada que se cierra a si misma (`<br />`). Este es el único nodo que puede tener tantos nodos hijos como atributos.
  - **Attr**. Este nodo representa el nombre del atributo o valor.

Objetos del modelo, propiedades y métodos de los objetos

- Objetos del modelo (3):
  - **Text**. Este nodo almacena la información que es contenida en el tipo de nodo `Element`.
  - **CDataSection**. Este nodo representa una secuencia de código del tipo `<![CDATA[ ]]>`. Este texto solo será analizado por un programa de análisis.
  - **Comment**. Este nodo representa un comentario XML.

Objetos del modelo, propiedades y métodos de los objetos

- La interfaz `Node`:
  - Para poder manipular la información de los nodos, JavaScript crea un objeto denominado `Node`.
  - En este objeto se definen las propiedades y los métodos para procesar los documentos.
  - Este objeto define una serie de constantes que identifican los tipos de nodo.

Objetos del modelo, propiedades y métodos de los objetos

- Constantes del objeto `Node`:

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

Objetos del modelo, propiedades y métodos de los objetos

- Métodos y propiedades de `Node` :

Propiedad / Método	
`nodeName`	`previousSibling`
`nodeValue`	`nextSibling`
`nodeType`	`hasChildNodes()`
`ownerDocument`	`attributes`
`firstChild`	`appendChild(nodo)`
`lastChild`	`removeChild(nodo)`
`childNodes`	`replaceChild(nuevoNodo, anteriorNodo)`
`parentNode`	`insertBefore(nuevoNodo, anteriorNodo)`

Acceso al documento desde código

- Cuando el árbol de nodos DOM ha sido construido por el navegador de forma automática, podemos acceder a cualquier nodo.
- Si existe más de un elemento, estos se van almacenando en un array.

Acceso al documento desde código

- Supongamos que tenemos una página HTML con la siguiente estructura:

```
<html>
  <head>
    <title>Titulo DOM</title>
  </head>
  <body>
    <p>Parrafo DOM</p>
    <p>Parrafo DOM segundo</p>
    <p>Parrafo DOM tres</p>
  </body>
</html>
```

Acceso al documento desde código

- El primer paso es recuperar el objeto que representa el elemento raíz de la página:

```
var obj_html = document.documentElement;
```

- De este elemento derivan `<head>` y `<body>`.

Acceso al documento desde código

- Podríamos acceder al primer y último hijo del nodo `<html>`:

```
var obj_head = obj_html.firstChild;
var obj_body = obj_html.lastChild;
```

Acceso al documento desde código

- Si existiesen más nodos podemos acceder a ellos a través del índice:

```
var obj_head = obj_html.childNodes[0];
var obj_body = obj_html.childNodes[1];
```

Acceso al documento desde código

- Si no supiésemos el número de nodos hijo, podemos acceder a este valor:

```
var numeroHijos = obj_html.childNodes.length;
```

Acceso al documento desde código

- Otra forma de acceder a un nodo es a través de su hijo:

```
var obj_html = obj_body.parentNode;
```

Acceso a los tipos de nodo

- Los objetos del modelo se diferencian por su tipo.
- La forma de acceder al tipo de nodo es mediante la propiedad `nodeType`.

```
obj_tipo_document = document.nodeType; // 9
obj_tipo_elemento = document.documentElement.nodeType; // 1
```

Acceso directo a los nodos

- DOM añade una serie de métodos para acceder de forma directa a los nodos:
  - `getElementByTagName()`.
  - `getElementsByName()`.
  - `getElementById()`.

Acceso directo a los nodos

- `getElementByTagName()` recupera los elementos de la página HTML de la etiqueta que hayamos pasado como parámetro.

```
var divs = document.getElementsByTagName("div");
```

Acceso directo a los nodos

- `getElementByName()` recupera todos los elementos de la página HTML en los que el atributo `name` coincide con el parámetro pasado a través de la función.

```
var divPrimero = document.getElementsByName("primero");

<div name="primero">...</div>
<dif name="segundo">...</div>
<div>...</div>
```

Acceso directo a los nodos

- `getElementByID()` recupera el elemento HTML cuyo ID coincida con el pasado a través de la función.

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

<div id="pie">
  <a href="URL" id="imagen">...</a>
</div>
```

Acceso a los atributos de un nodo tipo element

- DOM permite acceder directamente a todos los atributos de una etiqueta.
- La propiedad `attributes` permite acceder a los atributos de un nodo de tipo `element` mediante los siguientes métodos:
  - **getNameItem(nomAttr)**. Devuelve el nodo de tipo `attr` (atributo), cuya propiedad `nodeName` (Nombre del nodo) contenga el valor `nomAttr`.

Acceso a los atributos de un nodo tipo element

- **removeNameItem(nomAttr)**. Elimina el nodo de tipo `attr` (atributo) en el que la propiedad `nodeName` (Nombre del nodo) coincida con el valor `nomAttr`.
- **setNameItem(nodo)**. Este método añade el nodo `attr` (atributo) a la lista de atributos del nodo element**.** Lo indexa según la propiedad `nodeName` (del atributo).
- **item(pos)**. Devuelve el nodo correspondiente a la posición indicada por el valor numérico `pos`.

Acceso a los atributos de un nodo tipo element

- DOM proporciona algunos métodos que permiten un acceso directo a la modificación, inserción y borrado de los atributos de una etiqueta:
  - **getAttribute(nomAtributo)**. Este método equivale a: `attributes.getNameItem(nombAtributo)`.
  - **setAttribute(nomAtributo, valorAtributo)**. Este método equivale a la estructura: `attributes.getNamedItem(nomAtributo).value = valor`
  - **removeAttribute(nomAtributo)**. Este método equivale a la estructura: `attributes.removeNameItem(nomAtributo)`

Creación y eliminación de nodos

- DOM permite crear nodos en el árbol de forma dinámica a través de los siguientes métodos:

  - **createAttribute(nomAtributo)**. Este método crea un nodo de tipo atributo con el nombre pasado a la función.
  - **createCDataSection(textoPasado)**. Este método crea una sección de tipo `CDATA` con un nodo hijo de tipo texto con el valor `textoPasado`.
  - **createComent(textoPasado)**. Este método crea un nodo de tipo `comment` (comentario), con el contenido de `textoPasado`.
  - **createDocumentFragment()**. Este método crea un nodo de tipo `DocumentFragment`.

Creación y eliminación de nodos

- **createElement(nomEtiqueta)**. Este método crea un elemento del tipo etiqueta, del tipo del parámetro pasado como `nomEtiqueta`.
- **createEntityReference(nomNodo)**. Este método crea un nodo de tipo `EntityReference`.
- **createProcessingInstruction(objetivo,dato)**. Este método crea un nodo de tipo `ProcessingInstruction`.
- **createTextNode(textoPasado)**. Este método crea un nodo de tipo texto con el valor del parámetro pasado, `textoPasado`.

Programación de eventos

- Los eventos se utilizan para relacionar la interacción del usuario con las acciones de DOM vistas hasta ahora.

Programación de eventos

- Carga de la página HTML:
  - Una condición para que se genera la estructura de árbol es que la página se cargue completamente.
  - Por este motivo es necesario conocer si se ha cargado y para ello se utiliza el evento `onload`.

```
<html>
  <head><title>Titulo DOM</title></head>
  <body onload="alert('Página cargada completamente');">
    <p>Primer parrafo</p>
  </body>
</html>
```

Programación de eventos

- Comprobar si el árbol DOM está cargado:

```
<html>
  <head><title>Titulo DOM</title>
    <script>
      function cargada() {
        window.onload = "true";
        if(window.onload){ return true; }
        return false;
      }
      function pulsar(){
        if(cargada()){ alert("Página cargada correctamente");
        }
      }
    </script>
  </head> <body> <p onclick="pulsar();">Primer párrafo</p> </body>
</html>
```

Programación de eventos

- Actuar sobre el DOM al desencadenarse eventos:

```
<html>
  <head><title>Titulo DOM</title>
    <script>
       function ratonEncima(){
        document.getElementsByTagName("div")
          [0].childNodes[0].nodeValue="EL RATON ESTA ENCIMA";
      }
      function ratonFuera(){
        document.getElementsByTagName("div")
          [0].childNodes[0].nodeVale="NO ESTA EL RATON ENCIMA";
      }
    </script>
  </head>
  <body><div onmouseover="ratonEncima(); "onmouseout="ratonFuera();">
        VALOR POR DEFECTO</div></body></html>
```

Diferencias en las implementaciones del modelo

- Una de las principales dificultades a la hora de utilizar DOM es que no todos los navegadores hacen una misma interpretación.
- La guerra entre navegadores a la hora de generar sus propios estándares, ha generado muchos problemas a los programadores de páginas web.

Diferencias en las implementaciones del modelo

- Adaptaciones de código para diferentes navegadores:
  - Pueden ocasionar problemas de interpretación.
  - Complican la actualización futura de la página.
  - Generan la necesidad de modificar el código implementado.

Diferencias en las implementaciones del modelo

- Adaptaciones de código para diferentes navegadores:
  - Internet Explorer ha añadido su propia extensión de DOM, con lo cual genera problemas de interoperabilidad entre los navegadores web.

Diferencias en las implementaciones del modelo

- Adaptaciones que se pueden realizar para mejorar la compatibilidad:
  - Crear de forma explícita las constantes predefinidas.

```
alert(Node.DOCUMENT_NODE); // Devolvería 9
alert(Node.ELEMENT_NODE); // Devolvería 1
alert(Node.ATTRIBUTE_NODE); // Devolvería 2
```

Diferencias en las implementaciones del modelo

- Crear de forma explícita las constantes predefinidas:

```
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
  };
}
```