DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 6:

Utilización del modelos de objetos del documento

(DOM-Document Object Model)

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, y a sus elementos.
- 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.
- 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.
- En noviembre de 2015 se ha publicado **DOM nivel 4**

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.

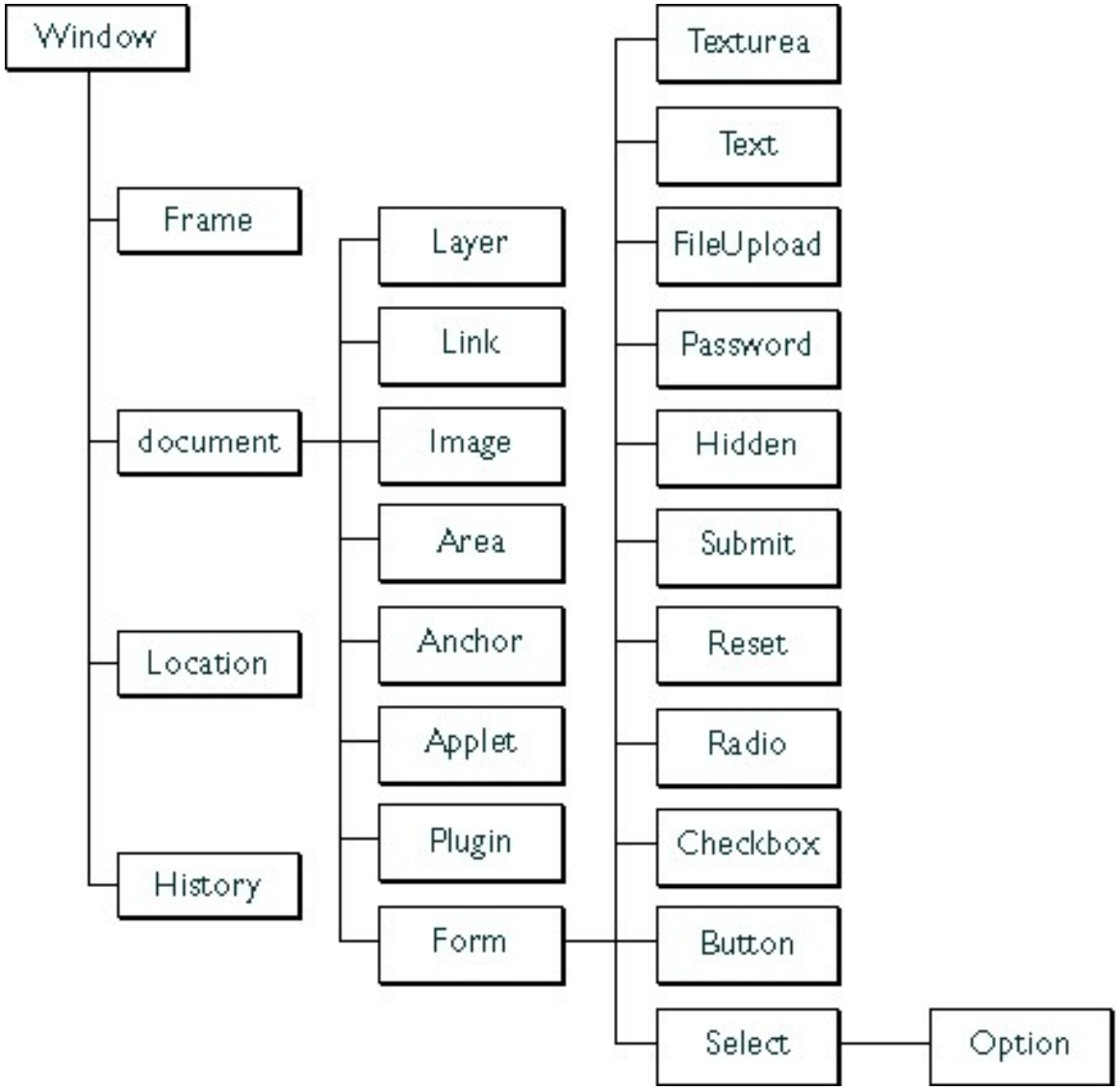
El modelo de objetos del documento (DOM)

- 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.

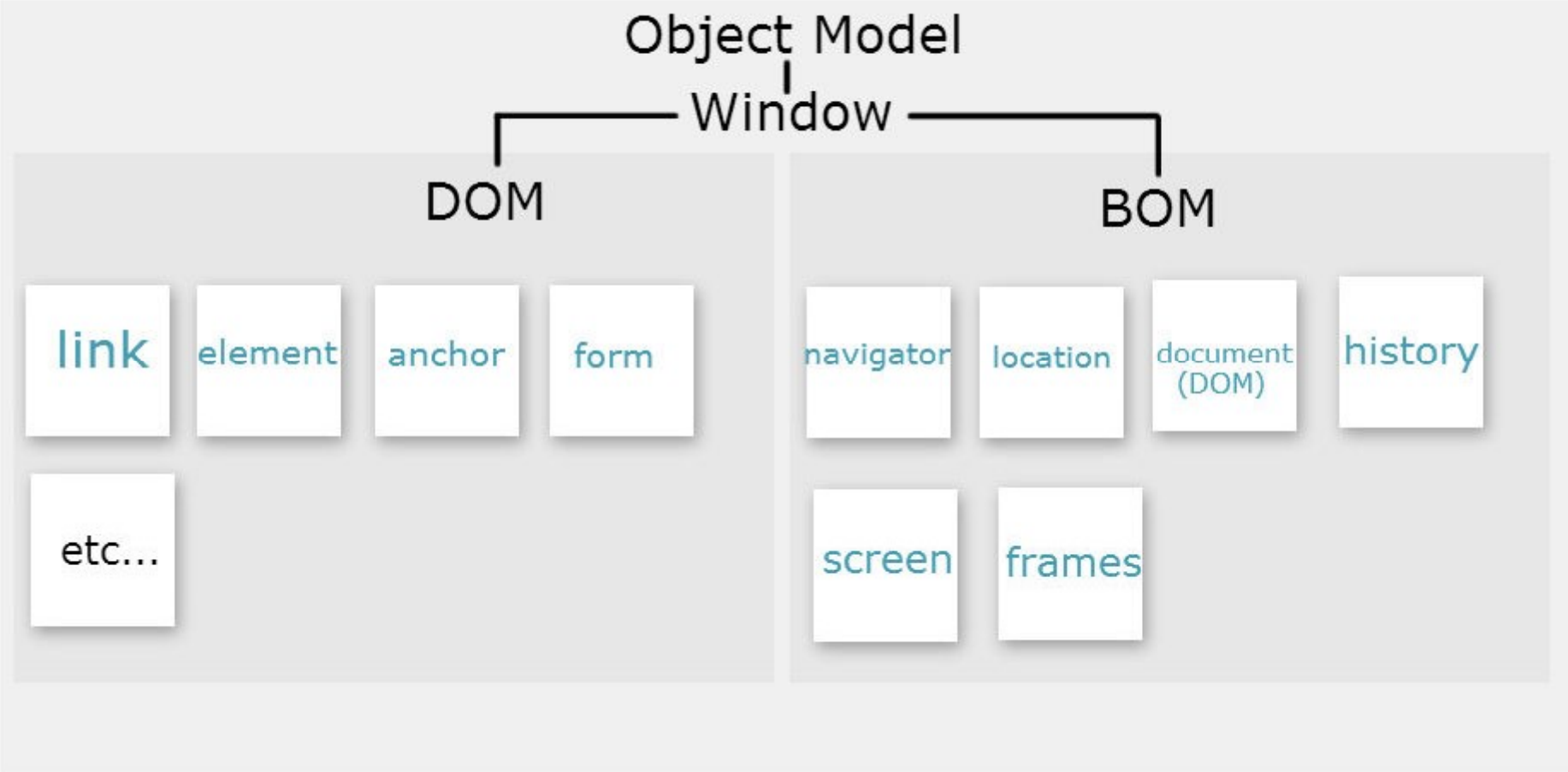
El modelo de objetos del documento (DOM)

- HTML DOM:
  - Un modelo de datos estándar para HTML.
  - Es un estándar de la W3C.
  - El DOM HTML define los objetos, las propiedades de los elementos HTML, los métodos para acceder a ellos y los eventos que se pueden producir.
  - **Es un estándar sobre la forma de obtener, modificar, añadir o eliminar elementos HTML.**

HTML DOM


Window
Frame
document
Location
History
Layer
Link
Image
Area
Anchor
Applet
Plugin
Form
Texturea
Text
FileUpload
Password
Hidden
Submit
Reset
Radio
Checkbox
Button
Select
Option

HTML DOM


Object Model
Window
DOM
BOM
link
element
anchor
form
etc...
navigator
location
document (DOM)
history
screen
frames

Acceso directo a objetos HTML DOM

- DOM añade una serie de métodos para acceder de forma directa a los elementos HTML:
  - `getElementsById(id).`
  - `getElementsByTagName(etiqueta).`
  - `getElementsByName(nombre).`
  - `getElementsByClassName(nombre).`
  - `querySelector(selector Css)`
  - `querySelectorAll(selector Css)`
  - Colecciones de objetos

Acceso directo a objetos HTML DOM

- **`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>
```

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

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

Acceso directo a objetos HTML DOM

- **`getElementsByClassName()`** recupera todos los elementos de la página HTML que sean de la clase que se pasa como parámetro:

```
let elMiclase=document.getElementsByClassName("miClase")[0]
```

```
<div   class="miClase">...</div>
<dif   class="miClase">...</div>
<div>...</div>
```

- **`getElementsByName()`** 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 objetos HTML DOM

- **`querySelector()`** Devuelve el primer elemento del documento que coincida con el grupo especificado de selectores.

```
var x= document.querySelector("p.intro");

<div id="pie">
  <p class= "intro">...</p>
</div>
```

- **`querySelectorAll()`** recupera una lista de todos los elementos HTML que coincidan con el selector pasado como argumento.

```
var x= document.querySelectorAll("p.intro");

<div id="pie">
  <p class= "intro">...</p>
</div>
```

Acceso directo a objetos HTML DOM

- Colecciones de objetos
  - document.anchors
  - document.body
  - document.documentElement
  - document.embeds
  - document.forms
  - document.head
  - document.images
  - document.links
  - document.scripts
  - document.title

Acceso a objetos HTML y Colecciones de objetos

```
1  <html>
2    <head><title>Pagina de ejemplo</title></head>
3    <body>
4      <p>Primer parrafo de la pagina</p>
5      <a href="otra_pagina.html">Un enlace</a>
6      <img src="logo.jpg" name="logotipo"/>
7      <form method="post" name="consultas">
8        <input type="text" name="id" />
9        <input type="submit" value="Enviar">
10     </form>
11   </body>
12 </html>
```

Párrafo: document.getElementsByTagName("p")
Enlace: document.links[0]
Imagen: document.images[0] o document.images["logotipo"]
Formulario: document.forms[0] o document.forms["consultas"]

Cambiar elementos HTML

- `Element.innerHTML = contenido html`
  Cambia el contenido de un elemento HTML

- `Element.attribute = nuevo valor`
  Cambia el valor de un atributo de un elemento HTML(excepto class)

- `Element.setAttribute(attribute,value)`
  Cambia el valor de un atributo de un elemento HTML

```
document.getElementById("parrafo1").setAttribute("class","miClase");
document.getElementById("parrafo2").className = "miClase";
```

- `Element.style.property = nuevo estilo`
  Cambia el estilo de un elemento HTML

Añadir y eliminar elementos HTML

- `document.createElement(element)`
  Crea un elemento HTML
- `document.removeChild(element)`
  Borra un elemento HTML
- `document.appendChild(element)`
  Añade un elemento HTML
- `document.replaceChild(element)`
  Cambia un elemento HTML por otro
- `document.write(text)`
  Escribe el contenido del documento HTML

HTML DOM

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3

HTML DOM

Property	Description	DOM
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1

HTML DOM

Property	Description	DOM
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

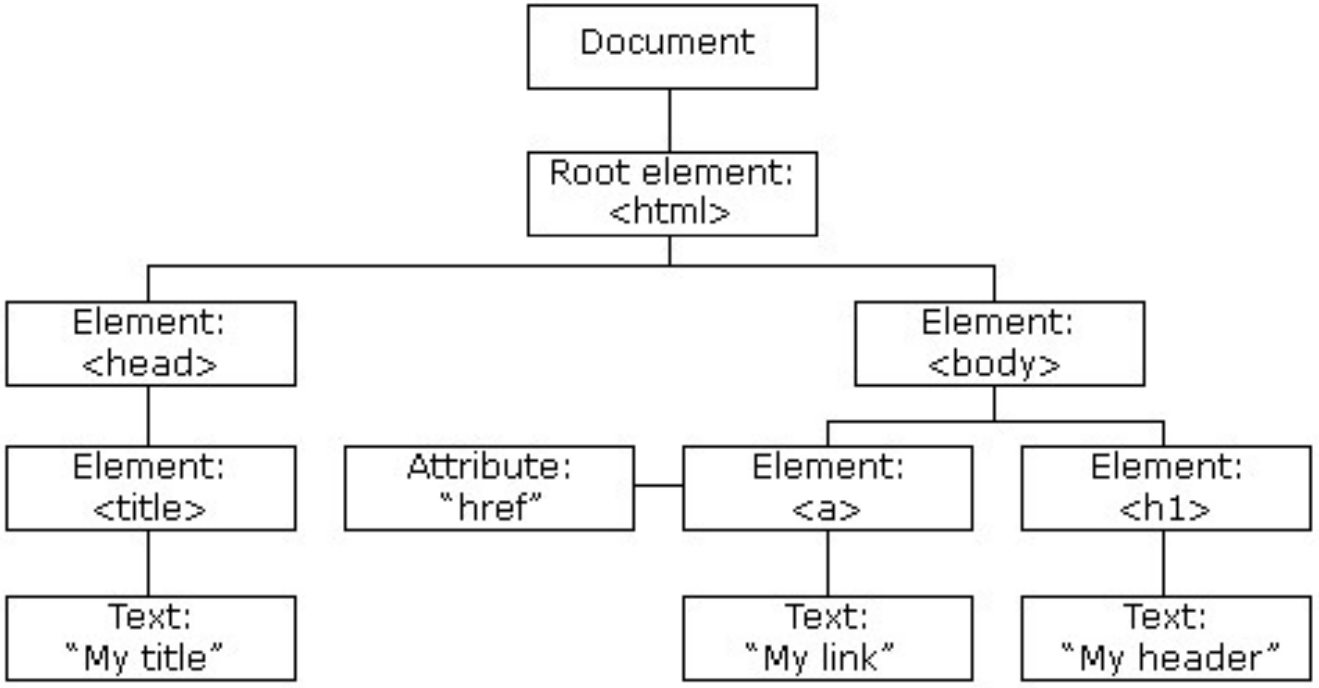
El modelo de objetos del documento (DOM)

- Estructura del árbol DOM:

```
<!DOCTYPE html>
<html>
        <head>
                <title>My title</title>
        </head>
        <body>
                <a href="">My link</a>
                <h1>My header</h1>
        </body>
</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 (html), 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.

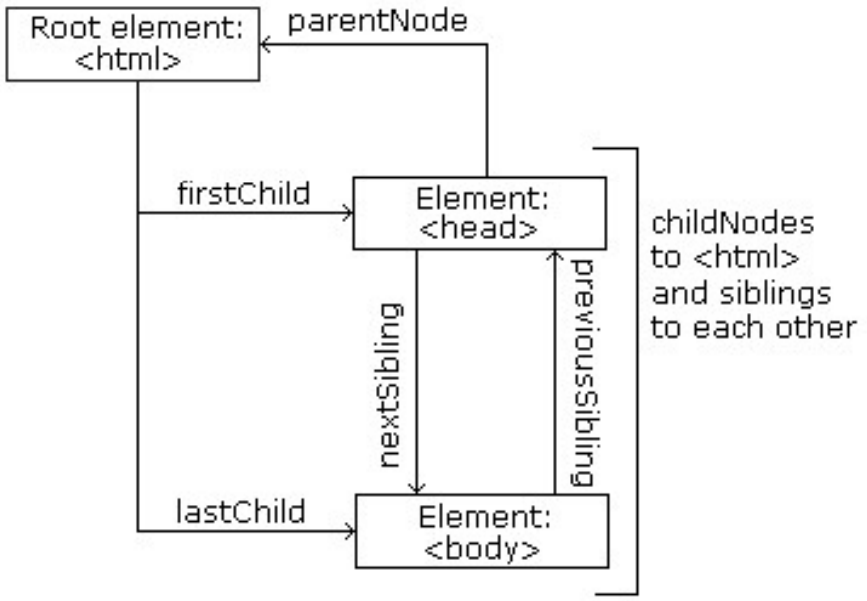
El modelo de objetos del documento (DOM)

```
<html>

  <head>
      <title>DOM Tutorial</title>
  </head>

  <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
  </body>

</html>
```



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 />`). Se trata del único nodo que puede contener atributos y del que pueden derivar otros nodos.
  - **Attr**. Este nodo representa el nombre del atributo o valor.

```
<p id="parrafo1">Este es el primer párrafo</p>
```

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.

```
<p id="parrafo1">Este es el primer párrafo</p>
```

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`	
`nodeValue`	
`nodeType`	`hasChildNodes()`
`ownerDocument`	`cloneNode(boolean)`
`firstChild`	`appendChild(nodo)`
`lastChild`	`removeChild(nodo)`
`childNodes`	`replaceChild(nuevoNodo, anteriorNodo)`
`parentNode`	`insertBefore(nuevoNodo, anteriorNodo)`
`previousSibling`	
`nextSibling`	
`attributes`	

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>`.

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

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

document.body.innerHTML → Accede al body del documento

document.documentElement → Accedemos al documento completo

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];
```

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

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

- 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 a los tipos de nodo

- Un error común es pensar que un nodo elemento contiene texto.

- Ejemplo:

  <title> DOM tutorial </title>

  El elemento <title> no contiene texto, contiene un nodo texto con el valor "DOM tutorial".

  El valor de un nodo texto se puede cambiar o acceder mediante la propiedad **innerHTML** o **nodeValue**

Acceso a los tipos de nodo

```
<html>
  <body>
     <h1 id="intro">My First Page</h1>
     <p id="demo">Hello!</p>
     <script>
      var myText = intro.childNodes[0].nodeValue;
      demo.innerHTML = myText;
     </script>

  </body>
</html>
```

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`.

Acceso a los atributos de un nodo tipo element

	nodeName	nodeValue	attributes
Element	tagName	null	NamedNodeMap
Attr	name of attribute	value of attribute	null
Text	#text	content of the text node	null
CDATASection	#cdata-section	content of the CDATA Section	null
EntityReference	name of entity referenced	null	null
Entity	entity name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentType	document type name	null	null
DocumentFragment	#document-fragment	null	null
Notation	notation name	null	null

Creación y eliminación de nodos

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" >
  </head>
  <body>

  <div id="div1">
    <p id="p1">Párrafo 1</p>
    <p id="p2">Párrafo 2</p>
  </div>

  <script>
  var para = document.createElement("p");
  var node = document.createTextNode("Párrafo nuevo");
  para.appendChild(node);
  var element = document.getElementById("div1");
  element.appendChild(para);
  </script>

  </body>
</html>
```

Creación y eliminación de nodos

```
<!DOCTYPE html>
<html lang="es">
        <head>
                <meta charset="UTF-8">
        </head>
        <body>

        <div id="div1">
                <p id="p1">Párrafo 1</p>
                <p id="p2">Párrafo 2</p>
        </div>

        <script>
                var para = document.createElement("p");
                var node = document.createTextNode("Párrafo nuevo");
                para.appendChild(node);

                div1.insertBefore(para,p1);
        </script>

        </body>
</html>
```

Creación y eliminación de nodos

```
<!DOCTYPE html>
<html lang="es">
        <head>
                <meta charset="UTF-8">
        </head>
        <body>

        <div id="div1">
                <p id="p1">Párrafo 1</p>
                <p id="p2">Párrafo 2</p>
        </div>

        <script>
                var para = document.createElement("p");
                var node = document.createTextNode("Párrafo nuevo");
                para.appendChild(node);

                div1.replaceChild(para,p1);
        </script>

        </body>
</html>
```

Creación y eliminación de nodos

```
<!DOCTYPE html>
<html lang="es">
      <head>
            <meta charset="UTF-8">
      </head>
      <body>

      <div id="div1">
            <p id="p1">Párrafo 1</p>
            <p id="p2">Párrafo 2</p>
      </div>

      <script>

            div1.removeChild(p1);
      </script>

      </body>
</html>
```

Creación y eliminación de nodos

```
<!DOCTYPE html>
<html>
<body>

<div style="border:1px solid black;background-color:pink">
  <p style="color:red;">A p element</p>
  <p style="color:green;">A p element</p>
  <p style="color:blue;">A p element</p>
</div>


<input type="button" onclick="myFunction()" value='Try
it'></input>

<script>
  function myFunction() {
    var elmnt = document.getElementsByTagName("DIV")[0];
    var cln = elmnt.cloneNode(true);
    document.body.appendChild(cln);
  }
</script>

</body>
</html>
```

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:
  - **getNamedItem(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

- **removeNamedItem(nomAttr)**. Elimina el nodo de tipo `attr` (atributo) en el que la propiedad `nodeName` (Nombre del nodo) coincida con el valor `nomAttr`.

- **setNamedItem(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.getNamedItem(nombAtributo).value`
  - **setAttribute(nomAtributo, valorAtributo)**. Modificar atributos existentes. Este método equivale a la estructura: `attributes.getNamedItem(nomAtributo).value = valor`

Acceso a los atributos de un nodo tipo element

- **setAttribute(nomAtributo, valorAtributo)**. – Añadir atributos. Este método equivale a la estructura:

```
atributo = document.createAttribute(nombreAtributo);
atributo.value = "valorAtributo";
attributes.setNamedItem(atributo);
```

- **hasAttribute(nomAtributo)**
- **removeAttribute(nomAtributo)**. Este método equivale a la estructura:

```
attributes.removeNamedItem(nomAtributo)
```

Generación dinámica de tablas

- Podemos acceder a una tabla que ya existe

```
var x = document.getElementById("myTable");
```

- O crear un objeto Table

```
var x = document.createElement("TABLE");
```

Generación dinámica de tablas

- **Colecciones del objeto Table**

Colección	Descripción
rows	Devuelve todos los elementos <tr> que hay en la tabla
tBodies	Devuelve todos los elementos <tbody> que hay en la tabla

Generación dinámica de tablas

- **Propiedades del objeto Table**

Propiedad	Descripción
Caption	Devuelve el elemento <caption> de la tabla
tFoot	Devuelve una referencia al elemento <tfoot> de la tabla
tHead	Devuelve una referencia al elemento <thead> de la tabla

Generación dinámica de tablas

- **Métodos del objeto Table**

Propiedad	Descripción
createCaption()	Crea un elemento <caption> vacío y lo añade a la tabla
createTFoot()	Crea un elemento <tfoot> vacío y lo añade a la tabla
createTHead()	Crea un elemento <thead> vacío y lo añade a la tabla
deleteCaption()	Elimina el primer elemento <caption> de la tabla
deleteRow()	Elimina una fila de la tabla
deleteTFoot()	Elimina el elemento <tfoot> de la tabla
deleteTHead()	Elimina el elemento <thead> de la tabla
insertRow()	Crear un elemento <tr> vacío y lo inserta en la tabla

Generación dinámica de tablas

- Podemos acceder a una fila que ya existe

```
var x = document.getElementById("myTr");
```

- O crear un objeto TableRow

```
var x = document.createElement("TR");
```

Generación dinámica de tablas

- **Colecciones del objeto TableRow**

Colección	Descripción
Cells	Devuelve todos los elementos <td> y <th> que hay en una fija de una tabla

Generación dinámica de tablas

- **Propiedades del objeto TableRow**

Propiedad	Descripción
rowIndex	Devuelve la posición de una fila de la colección rows de una tabla
sectionRowIndex	Devuelve la posición de una fila de la colección rows de un tbody, thead o tfoot

Generación dinámica de tablas

- **Métodos del objeto Tabla**

Propiedad	Descripción
deleteCell()	Elimina una celda de la fila de la tabla
insertCell()	Inserta una celda en la fila de la tabla

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].nodeValue="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.
  - 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
  };
}
```

Uso de librerías de terceros

- Para solucionar el problema del desarrollo de aplicaciones web en diferentes navegadores nace *cross-browser*.
- *Cross-browser* tiene la intención de visualizar una página o aplicación web exactamente igual en todos los navegadores.
- Para conseguir este objetivo surgen utilidades que permiten unificar los eventos y sus propiedades.

Uso de librerías de terceros

- **Renderizar a través de una Web:**
  - Existen páginas Web que nos permiten introducir una dirección de una página Web y elegir la versión del navegador con el que queremos visualizarlo.
  - Por ejemplo, *netrenderer* nos permite visualizar una página Web en las distintas versiones de Internet Explorer.
  - Normalmente este tipo de páginas, solo nos muestran una imagen del resultado, por lo que puede resultar complicado en ocasiones solucionar algunos problemas.

Uso de librerías de terceros

- **Programas para renderizar:**
  - Existen programas que nos permiten instalar varias versiones del mismo navegador, como *multi IE* o *Internet Explorer Collection*.
  - Estos programas dan problemas de compatibilidad de versiones con los últimos navegadores.

Uso de librerías de terceros

- **Instalar los navegadores en máquinas virtuales.**
  - Otra opción es instalar las versiones de navegadores en máquinas virtuales, que además estén acorde con los sistemas operativos para los que hay instalables de la versión de navegador.

Uso de librerías de terceros

- Para el uso de las utilidades *cross-browser* es necesario implementar funciones que habitualmente vienen definidas en librerías.
- Es necesario comprobar el navegador y tomar una decisión u otra en función de la respuesta.

```
if (window.XMLHttpRequest){
  // codigo para IE7+, Firefox, Chrome, Opera, Safari
  Xmlhttp = new XMLHttpRequest();
}else{
  // codigo para IE6, IE5
  Xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```