

DESARROLLO WEB EN ENTORNO CLIENTE

U.D. 5:

**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.
- Es una interfaz de programación de aplicaciones (API) de la plataforma de W3C independiente de cualquier lenguaje de programación.
- 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 v2.0. A esta primera versión de DOM se le denomina DOM nivel 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.

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.

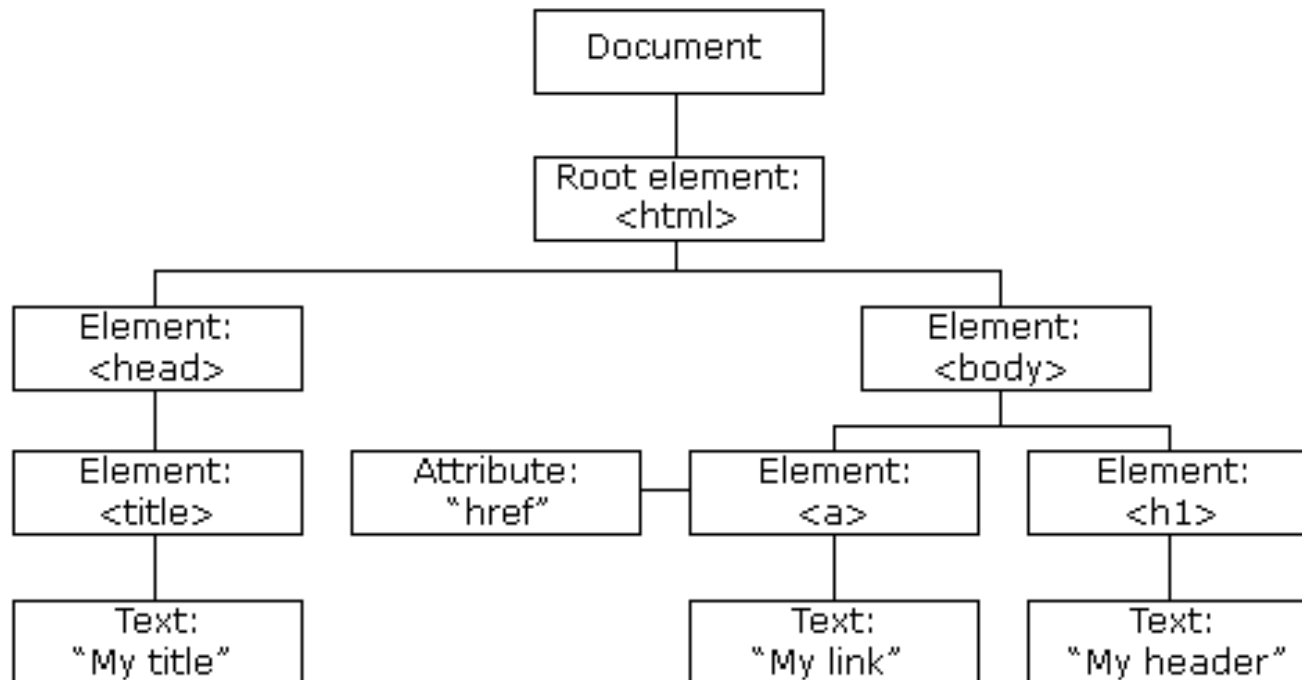
El modelo de objetos del documento (DOM)

■ 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)

- Código HTML de la estructura del árbol DOM anterior:

```
<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)

- Para ordenar la estructura del árbol, existe una serie de reglas:
 - En el árbol de nodos, al nodo superior (document), se le llama raíz (root).
 - 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

- Tipos de nodos (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

- Tipos de nodos (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 (`
`). 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

- Tipos de nodos (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 / objeto `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.
 - Su propiedad `nodeType` define una serie de constantes que identifican los tipos de nodo.

Objetos del modelo, propiedades y métodos de los objetos

- Métodos y propiedades del objeto 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)

Objetos del modelo, propiedades y métodos de los objetos

- Asignación de valores al tipo de nodo (propiedad `nodeType` del objeto `Node`):

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

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>Título DOM</title>
  </head>
  <body>
    <p>Párrafo DOM primero</p>
    <p>Párrafo DOM segundo</p>
    <p>Párrafo DOM tercero</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

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

```
var obj_head = obj_html.firstChild;//obj nodo tipo <head>  
var obj_body = obj_html.lastChild;//obj nodo tipo <body>
```

- Si existen 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 sabemos el número de nodos hijo, podemos acceder a este valor:

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

- Cuando el árbol es complejo, una forma mejor de acceder a un nodo es a través de su hijo:

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

- El nombre de los `element` coincide con el de su etiqueta, se puede asignar el elemento al objeto:

```
var obj_head = document.head; // Elemento head  
var obj_body = document.body; // Elemento body
```

Acceso a los tipos de nodo

- Los objetos del modelo se diferencian por su tipo (en total son 12).
- 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:
 - `getElementsByTagName()`
 - `getElementByName()`
 - `getElementById()`
 - `getElementsByClassName()`

Acceso directo a los nodos

- `getElementsByName()`. Este método recupera los elementos de la página HTML de la etiqueta que hayamos pasado como parámetro devolviendo un vector con los nodos (una lista de nodos).

```
var divs = document.getElementsByTagName("div");  
  
for (var i=0; i<divs.length; i++) {  
    var div = divs[i];  
}
```

Acceso directo a los nodos

- `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>
```

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

```
<div>...</div>
```

Acceso directo a los nodos

- `getElementById()`. Recupera el elemento HTML cuyo atributo `id` coincida con el parámetro pasado a través de la función.

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

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

Acceso directo a los nodos

- `getElementsByClassName()`. Recupera todos los elementos de la página HTML en los que el atributo `class` coincide con el parámetro pasado a través de la función.

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

```
<div class="primero">...</div>
```

```
<div class="segundo">...</div>
```

```
<div>...</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 (no se suelen usar por complejidad):
 - **`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`

- **`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)`**. 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 la estructura:
`attributes.getNameItem(nombAtributo)`.
 - **`setAttribute(nomAtributo, valorAtributo)`**. Equivale a:
`attributes.getNamedItem(nomAtributo).value = valor`
 - **`removeAttribute(nomAtributo)`**. Equivale a:
`attributes.removeNameItem(nomAtributo)`

Acceso a los atributos de un nodo tipo element

- Código HTML de lo anterior:

```
<p id="parrafo" style="color:blue">Prueba de texto</p>
<script>
    var p = document.getElementById("parrafo");
    alert("p.id: " + p.id);

    // La variable "valorId" es "parrafo".
    var valorId = p.getAttribute("id");
    alert("valorId: " + valorId);

    // Se establece en atributo "id" valor "parrafoTexto".
    p.setAttribute("id", "parrafoTexto");
    alert("p.id: " + p.id);
</script>
```

- Con estos métodos podemos manejar todos los atributos que tiene una etiqueta 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)**. Crea una sección de tipo CDATA con un nodo hijo de tipo texto con el valor `textoPasado`.
 - **createComent(textoPasado)**. Crea un nodo de tipo comment (comentario), con el contenido de `textoPasado`.
 - **createDocumentFragment()**. Crea un nodo de tipo DocumentFragment.

Creación y eliminación de nodos

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

A través de estos métodos se puede modificar la estructura de la página para conseguir dinamismo total en el lado cliente con código limpio y ordenado.

Creación y eliminación de nodos

- Ejercicio: Crea un nodo en el árbol y añádele un valor.
- Ejercicio: Elimina el nodo en el árbol creado anteriormente.
- Ejercicio: Modifica el nodo en el árbol creado anteriormente.
- Ejercicio: Introduce un nuevo nodo en el árbol creado anteriormente antes del existente.

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 manejador de evento `onload`.

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

Programación de eventos

- Comprobar si el árbol DOM está cargado:

```
<html>
  <head><title>Árbol DOM cargado</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>DOM con evento</title>
  <script>
    function ratonEncima() {
      document.getElementsByTagName("div")[0].
        childNodes[0].nodeValue = "EL RATÓN ESTÁ ENCIMA";
    }

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

Gestor de eventos

- En JavaScript es posible añadir un gestor de eventos a un elemento especificado
- **addEventListener()**: permite asociar una función *callback* a un evento que se produzca sobre el elemento

```
elemento.addEventListener("evento", funcion);
```

```
var b1 = document.getElementById("button1");  
b1.addEventListener("click", pulsarBoton);
```

- **removeEventListener()**: se elimina un EventListener ya creado

```
elemento.removeEventListener("evento", funcion);
```

- Se pueden asociar EventListeners a cualquier objeto del DOM (ya sean elementos HTML, el objeto ventana, el documento...)

```
document.addEventListener("DOMContentLoaded", load);
```


Gestor de eventos

- Un elemento puede tener varios EventListeners asociados
 - Puede tener varios EventListeners asociados al mismo evento

```
<body>
  <button id="button1" onclick="funcion1()">Botón</button>
  <script>
    var b1 = document.getElementById("button1");
    b1.addEventListener("click", funcion2);
    b1.addEventListener("click", funcion3);
    b1.addEventListener("mouseover", funcion4);
    b1.addEventListener("click", funcion5);
    b1.removeEventListener("click", funcion2);

    function funcion1(){
      alert("Dentro de funcion1()");
    }

    function funcion2(){
      console.log("Dentro de funcion2()");
    }

    function funcion3(){
      console.log("Dentro de funcion3()");
    }

    function funcion4(){
      console.log("Dentro de funcion4()");
    }

    function funcion5(){
      console.log("Dentro de funcion5()");
    }
  </script>
</body>
```

Propiedad innerHTML

- Creado por Microsoft para Internet Explorer como método de acceso rápido al contenido de un contenedor en HTML. Recomendado por HTML5.
- Devuelve la sintaxis HTML describiendo los descendientes del elemento. Al establecerse se reemplaza la sintaxis HTML del elemento por la nueva.
 - Como lectura, permite por ejemplo obtener un string con lo que hay dentro de un div.
 - También permite escribir dentro y cambiar su contenido.

Propiedad innerHTML

- Ejemplo: Crea una página Web que muestre un texto y al pulsar sobre un botón se modifique haciendo uso de la propiedad *innerHTML*.

Ejercicios

- Realiza todos los ejercicios del bloque 'Ejercicios 5.1'.
- Realiza todos los ejercicios del bloque 'Ejercicios 5.2'.

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, provocó 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 que se pueden realizar para mejorar la compatibilidad:
 - Usar manejadores semánticos (asignar un identificador único al elemento XHTML mediante el atributo *id*, crear la función encargada de manejar el evento y asignarla al evento correspondiente en el elemento deseado).
 - 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 nació *cross-browser*.
- *Cross-browser* tiene la intención de visualizar una página o aplicación Web exactamente igual en todos los navegadores.
- Para conseguir este objetivo surgieron técnicas y 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, *BrowserShots* permite visualizar una página Web en distintos navegadores.
 - 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 o add-ons que permiten instalar varias versiones del mismo navegador.

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

Ejercicios

- Realiza todos los ejercicios del bloque 'Ejercicios 5.3'.