DOM

Juan Sevillano Hernández
2º DAW – Desarrollo Web Entorno Cliente

“

-1. Objetivos

-1. Objetivos

- Conocimientos para acceder a elementos DOM.
- Funciones para crear/eliminar elementos DOM.
- Ser capaz de ampliar el conjunto de instrucciones usadas mediante la visualización de ejemplos de código.
- Ser capaz de investigar en manuales o Internet para ampliar la base de instrucciones dadas en el tema.

“

00. Índice

ÍNDICE

1. Introducción.
2. Árbol de Nodos.
3. Tipos de Nodos.
4. Propiedades de Navegación entre los Nodos de tipo Elemento.
5. Funciones de JavaScript para localizar elementos en el DOM.
6. Funciones para Crear/Eliminar nodos.
7. Profundizar Más.
8. Agradecimiento.

“

1. Introducción

1. Introducción.

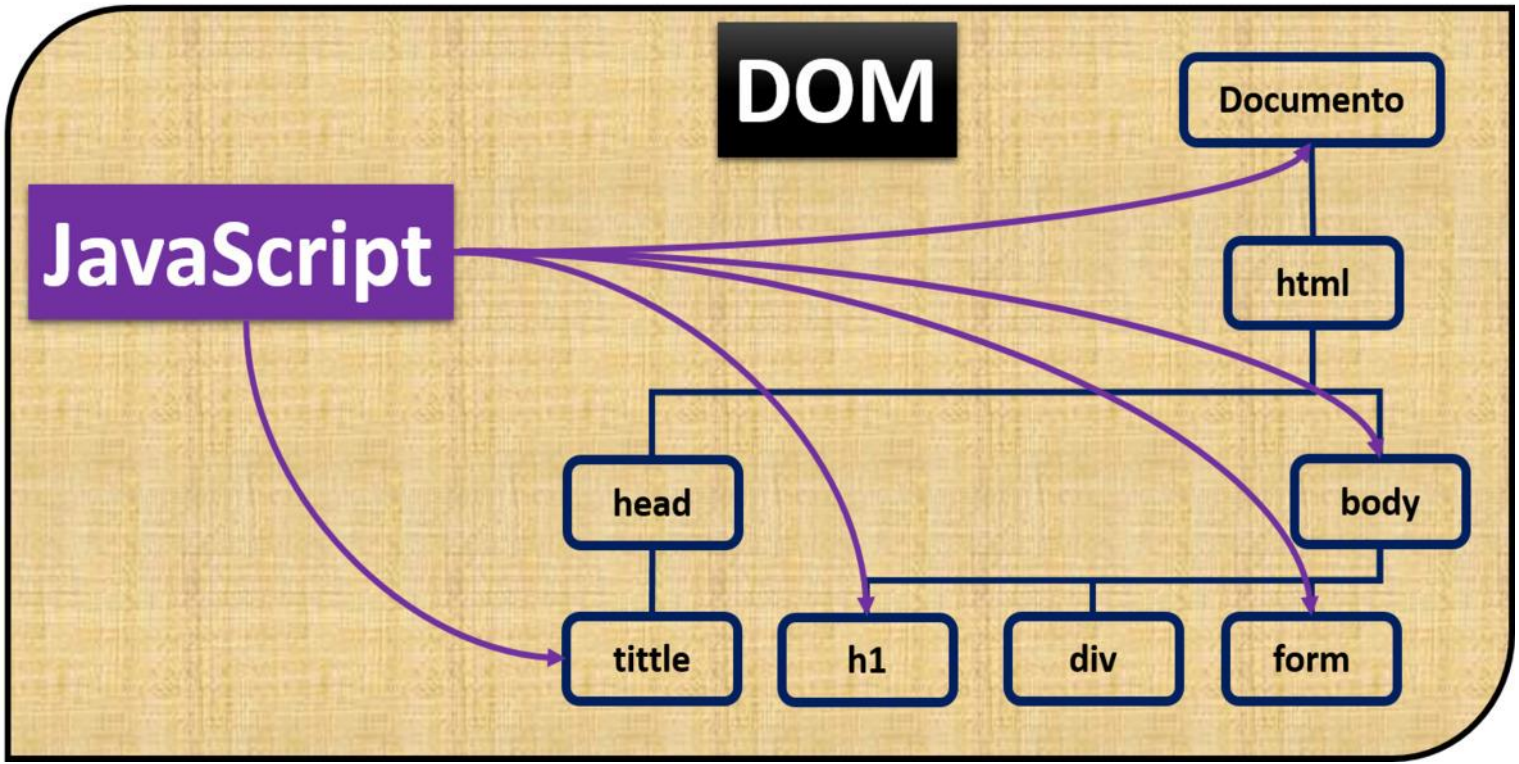
La creación del **Document Object Model** o **DOM** es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

¿Qué es DOM (Document Object Model)?

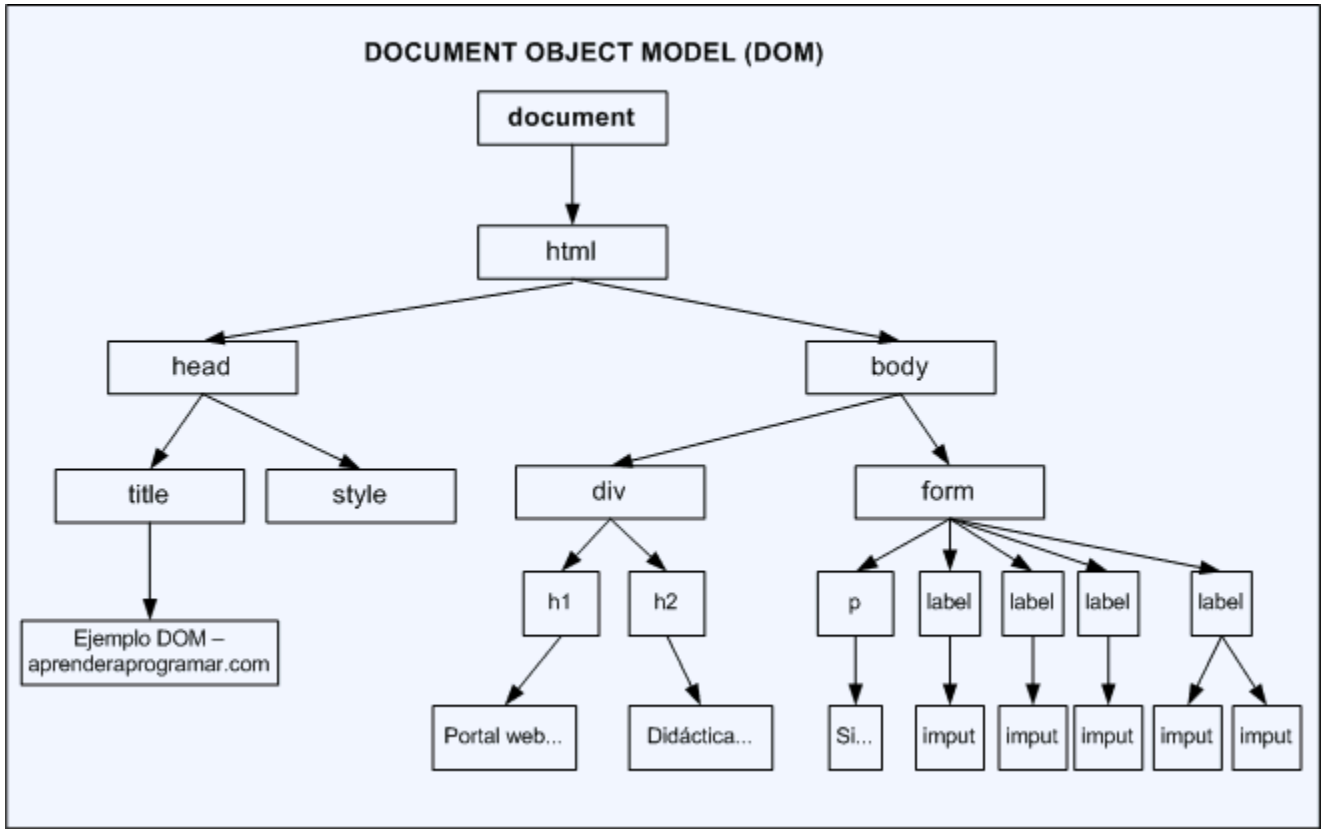
DOM es un modelo que permite tratar un documento Web XHTML (Marcado HTML) como si fuera un XML, navegando por los nodos existentes que forman la página web, pudiendo manipular sus atributos e incluso crear nuevos elementos.

Usando Javascript para navegar en el DOM podemos acceder a todos los elementos XHTML de la página web. Esto nos permite cambiar dinámicamente el aspecto de nuestra página web.

1. Introducción.



1. Introducción.



“

2. Árbol de Nodos.

2. Árbol de Nodos.

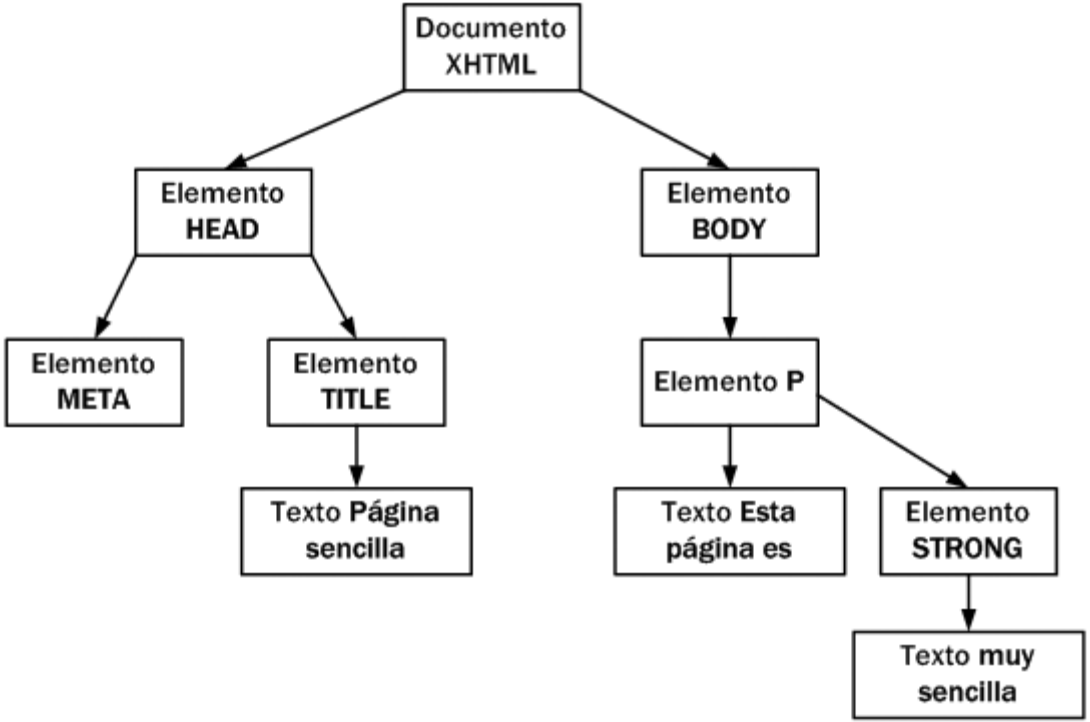
DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama “árbol de nodos”. La siguiente página web XHTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Página sencilla</title>
</head>

<body>
    <p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

2. Árbol de Nodos.

Se transforma en el siguiente árbol de nodos:



2. Árbol de Nodos.

La transformación automática de la página en un árbol de nodos siempre sigue la mismas reglas:

- Las etiquetas XHTML se transforman en dos nodos: el primero e la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta XHTML se encuentra dentro de la otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

“

3. Tipos de Nodos.

3. Tipos de Nodos.

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
- **Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
- **Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.
- **Comment**, representa los comentarios incluidos en la página XHTML.

4. Propiedades de Navegación entre los Nodos de Tipo Elemento.

4. Propiedades de Navegación entre los Nodos de Tipo Elemento.

Cualquier nodo de tipo elemento dispone de las siguientes propiedades para obtener la referencia de:

- **firstElementChild:** Almacena la referencia del primer nodo de tipo elemento hijo.
- **parentElement:** Almacena la referencia del nodo elemento padre.
- **lastElementChild:** Almacena la referencia del último nodo hijo.
- **nextElementSibling:** Almacena la referencia del siguiente nodo de tipo elemento (si lo tiene)
- **previousElementSibling:** Almacena la referencia del nodo de tipo elemento previo (si lo tiene)
- **children:** Almacena una colección con todos los elementos tipo nodo hijo.

“

5. Funciones de Javascript para localizar elementos en el DOM.

5. Función de Javascript para localizar elementos en el DOM.

Una vez construido automáticamente el árbol completo de nodos DOM, con las distintas funciones DOM podemos acceder de forma directa a cualquier nodo del árbol. El acceder a un nodo del árbol es equivalente a acceder a “un trozo” de la página web, siendo posible manipular de forma sencilla la página web, entre ello: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre (consisten en 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) y acceso directo (acceder directamente a ese nodo).

Nosotros vamos a trabajar solamente con el acceso directo al nodo ya que es menos costoso y nos ahorrará mucho tiempo-código programado.

5. Función de Javascript para localizar elementos en el DOM.

getElementsByTagName(nombreEtiqueta)

La función **getElementsByTagName(nombreEtiqueta) 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**. El valor devuelto es un **array de nodos DOM**, no un array de cadenas de texto o un array de objetos normales.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera: var primerParrafo = parrafos[0];

5. Función de Javascript para localizar elementos en el DOM.

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
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");
```

5. Función de Javascript para localizar elementos en el DOM.

getElementsByName(nombre)

La función **getElementsByName(nombre) es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo name sea igual al parámetro proporcionado.**

En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

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

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

5. Función de Javascript para localizar elementos en el DOM.

Normalmente el atributo **name** es único para los elementos HTML que lo definen, 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.

5. Función de Javascript para localizar elementos en el DOM.

getElementById(identificador)

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

La función **getElementById(identificador) 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>
```

5. Función de Javascript para localizar elementos en el DOM.

querySelector(identificadorElemento)

La función **querySelector(identificadorElemento) acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. En el caso de esta función, únicamente es devuelto el primer elemento que cumple la condición. Si no existe el elemento, el valor retornado es null.**

```
var logo = document.querySelector(".enlace");
```

```
<div id="cabecera">
  <a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
   <p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

5. Función de Javascript para localizar elementos en el DOM.

querySelectorAll(identificadorElemento)

La función **querySelectorAll(identificadorElemento) acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. Esta función devuelve un objeto de tipo NodeList con los elementos que coincidan con el selector.**

```
var enlaces = document.querySelectorAll(".enlace");
```

```
<div id="cabecera">
  <a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
   <p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

5. Función de Javascript para localizar elementos en el DOM.

Para acceder a los elementos almacenados en **NodeList**, recorremos el objeto como si de un array se tratase:

```
for (var i=0; i<enlaces.length; i++) {
   var enlaces = enlaces[i];
}
```

“

6. Funciones para Crear/Eliminar Nodos.

6. Funciones para Crear/Eliminar Nodos.

Creación de Elementos XHTML Simples.

Un elemento XHTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo **Element** y representa la etiqueta **<p>** y el segundo nodo es de tipo **Text** y representa el contenido textual de la etiqueta **<p>.**

Crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

1. Creación de un nodo de tipo **Element** que represente al elemento.
2. Creación de un nodo de tipo **Text** que represente el contenido del elemento.
3. Añadir el nodo **Text** como nodo hijo del nodo **Element**.
4. Añadir el nodo **Element** a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

6. Funciones para Crear/Eliminar Nodos.

De este modo, si se quiere añadir un párrafo simple al final de una página XHTML, es necesario incluir el siguiente código JavaScript:

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

6. Funciones para Crear/Eliminar Nodos.

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

- **createElement(etiqueta):** Crea un nodo de tipo **Element** que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- **createTextNode(contenido):** Crea un nodo de tipo **Text** que almacena el contenido textual de los elementos XHTML.
- **nodoPadre.appendChild(nodoHijo):** Añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo **Text** como hijo del nodo **Element** y a continuación se añade el nodo **Element** como hijo de algún nodo de la página.

6. Funciones para Crear/Eliminar Nodos.

Eliminación de Nodos.

Eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. En este caso, solamente es necesario utilizar la función **removeChild():**

Ejemplo:

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

var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
```

Resultado:

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

6. Funciones para Crear/Eliminar Nodos.

La función **removeChild()** requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser ejecutada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad **nodoHijo.parentNode.**

Así, para eliminar un nodo de una página XHTML se llama a la función **removeChild()** desde el valor **parentNode** del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

“

Profundizar Más.

Profundizar Más.

Si queréis profundizar más aún en todo lo relacionado con el DOM os dejo el siguiente enlace:

- https://www.tutorialesprogramacionya.com/herramientas/javascript/domjs/

“

Agradecimientos.

Agradecimientos.

Apuntes actualizados y adaptados para el CFGS DAW a partir de la siguiente documentación:

[1] Javascript Mozilla Developer.

[2] Javascript ES6 W3C.

[3] Apuntes Sergio García. CEEDCV.

“

¿Alguna Pregunta?

EJERCICIOS

Ejercicios DOM

CFGS 2º DAW

Curso: 2022/2023

Departamento de Informática

Juan Sevillano Hernández

Ejercicios: