DOM

¿Qué es?

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.

DOM permite a los programadores web **acceder y manipular las páginas HTML como si fueran documentos XML**. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A continuación se hace una breve descripción de un documento XML (se vera mas ampliado en el tema siguente.

- *Un **documento XML** es un fichero de texto en el que la información viene etiquetada.*
- *Está compuesto por un conjunto de elementos definidos en la recomendación XML de W3C.*
- *Puede ser leído por personas (bloc de notas o navegador) o por sistemas que extraen información.*
- *Para que lo lea un ordenador, el documento debe cumplir unas normas para que la aplicación pueda interpretarlo.*
- *Los documentos que cumplen estas normas se denominan documentos bien formados.* ○ *Documento XML: datos + etiquetas*

A pesar de sus orígenes, DOM se ha convertido en una utilidad **disponible para la mayoría de lenguajes de programación** (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

Árbol de nodos

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la **manipulación de las páginas web**. De esta forma, es habitual obtener el **valor almacenado por algunos elementos** (por ejemplo los elementos de un **formulario**), **crear un elemento** (párrafos, , etc.) de forma dinámica y añadirlo a la página, aplicar una **animación** a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy **sencillas de realizar gracias a DOM**. Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página original. Una **página HTML** normal no es más que una **sucesión de caracteres**, por lo que es un formato muy difícil de manipular. Por ello, los **navegadores web transforman** automáticamente todas las páginas web en una **estructura más eficiente de manipular**.

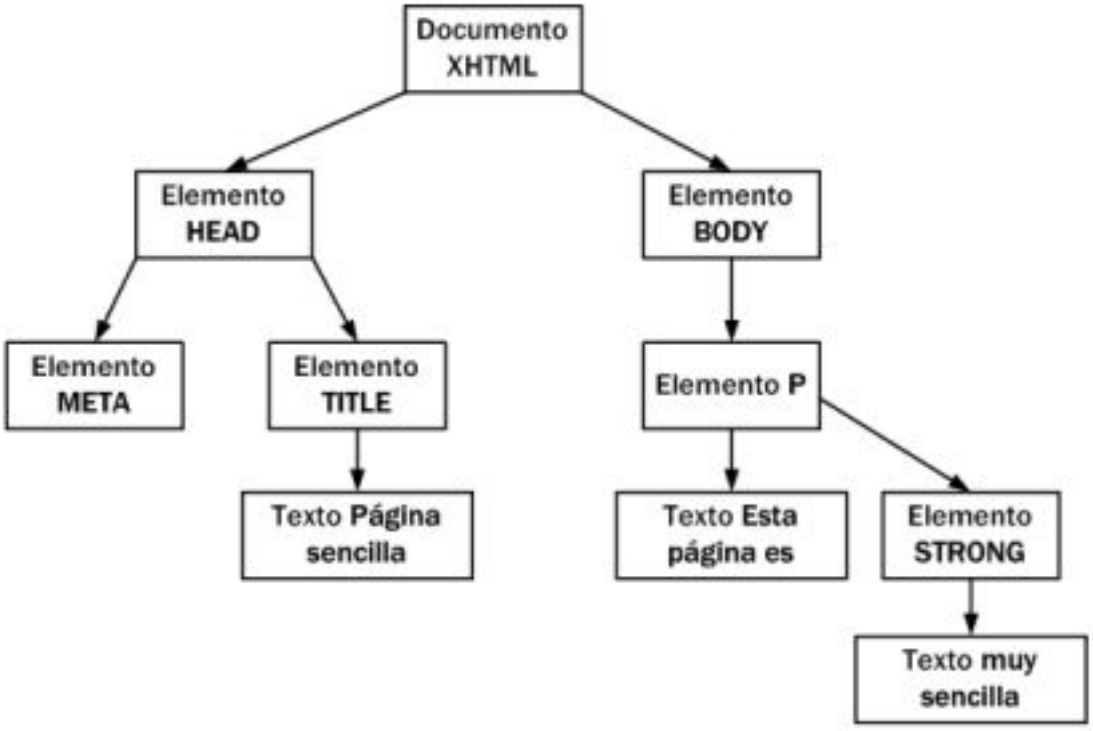
Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos HTML 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 HTML** sencilla:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se **transforma** en el siguiente **árbol de nodos**:



En el esquema anterior, cada **rectángulo** representa un **nodo DOM** y las **flechas** indican las **relaciones entre nodos**. Dentro de **cada nodo**, se ha incluido su **tipo** (que se verá más adelante) y su **contenido**.

La **raíz del árbol de nodos** de cualquier página HTML siempre es la misma: un **nodo** de tipo **especial** denominado **"Documento"**.

A partir de ese nodo raíz, **cada etiqueta HTML** se transforma en un **nodo** de tipo **"Elemento"**. La conversión de etiquetas en nodos se realiza **de forma jerárquica**. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada **etiqueta HTML** se transforma en un **nodo que deriva del nodo** correspondiente a su **"etiqueta padre"**.

La transformación de las **etiquetas HTML habituales** genera dos nodos: • nodo de tipo **"Elemento"** (correspondiente a la propia **etiqueta HTML**). • nodo de tipo **"Texto"** que contiene el **texto encerrado por esa etiqueta** HTML.

Así, la siguiente etiqueta HTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:

De la misma forma, la siguiente etiqueta HTML:

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

Genera los siguientes nodos:

- Nodo de tipo **"Elemento"** correspondiente a la etiqueta `<p>`.
- Nodo de tipo **"Texto"** con el contenido textual de la etiqueta `<p>`.
- Como el contenido de `<p>` incluye en su interior otra etiqueta HTML, la etiqueta interior se transforma en un nodo de tipo **"Elemento"** que representa la etiqueta `<strong>` y que deriva del nodo anterior.
- El contenido de la etiqueta `<strong>` genera a su vez otro nodo de tipo **"Texto"** que deriva del nodo generado por `<strong>`.

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

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

Como se puede suponer, las páginas HTML habituales producen árboles con miles de nodos. Aun así, el proceso de **transformación** es **rápido** y **automático**, siendo las **funciones proporcionadas por DOM** (que se verán más adelante) las únicas que permiten **acceder a cualquier nodo** de la página de forma sencilla e inmediata.

Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas HTML 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 HTML**. 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 HTML**, es decir, uno por cada par atributo=valor.
• **Text**, nodo que contiene el **texto encerrado por una etiqueta HTML**. • **Comment**, representa los **comentarios** incluidos en la página **HTML**.

Los otros tipos de nodos existentes que no se van a considerar son **DocumentType**, **CDataSection**, **DocumentFragment**, **Entity**, **EntityReference**, **ProcessingInstruction** y **Notation**.

Acceso directo a los nodos

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las **funciones DOM para acceder de forma directa a cualquier nodo del árbol.** Como acceder a un nodo del árbol es equivalente a **acceder a "un trozo" de la página**, una vez construido el árbol, ya es posible **manipular de forma sencilla la página**: 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**.
- Acceso **directo**.

Las funciones que proporciona DOM para acceder a un nodo 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. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho **más rápido acceder directamente a ese nodo** y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos.

Por último, es importante recordar que el **acceso a los nodos, su modificación y su eliminación** solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, **después de que la página HTML se cargue por completo**. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página HTML.

getElementsByTagName()

Como sucede con todas las funciones que proporciona DOM, la función `getElementsByTagName()` tiene un nombre muy largo, pero que lo hace **autoexplicativo**.

La función `getElementsByTagName(nombreEtiqueta)` obtiene todos los **elementos** de la **página** HTML 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 HTML:

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

El valor que se indica **delante** del nombre **de la función** (en este caso, `document`) es el **nodo a partir del cual se realiza la búsqueda de los elementos**. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor `document` como punto de partida de la búsqueda.

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**. Por lo tanto, **se debe procesar cada valor del array** de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener **el primer párrafo** de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

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");
```

getElementsByName()

La función `getElementsByName()` 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>
```

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.

getElementById()

La función `getElementById()` **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()` devuelve el **elemento HTML 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>
```

Creación y eliminación de nodos

Acceder a los nodos y a sus propiedades (que se verá más adelante) es sólo una parte de las manipulaciones habituales en las páginas. Las otras **operaciones habituales** son las de **crear** y **eliminar nodos** del árbol DOM, es decir, crear y eliminar "trozos" de la página web.

Creación de elementos HTML simples

Como se ha visto, un **elemento HTML 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>`.

Por este motivo, **crear y añadir** a la página un **nuevo elemento HTML** sencillo consta de **cuatro pasos** diferentes:

- **Creación de un nodo de tipo Element** que represente al **elemento**.
- **Creación de un nodo de tipo Text** que represente el **contenido** del **elemento**.
- **Añadir el nodo Text** como nodo **hijo del nodo Element**.
- **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.

De este modo, si se quiere añadir un párrafo simple al final de una página HTML, 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);
```

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 HTML cuya **etiqueta** se pasa como **parámetro**.
- **createTextNode(contenido)**: **crea** un **nodo** de tipo **Text** que almacena el contenido textual de los elementos HTML.
- **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**
  - A continuación se añade el **nodo Element** como **hijo** de algún **nodo** de la **página**.

Eliminación de nodos

Afortunadamente, **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()`:

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

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

La función `removeChild()` requiere como **parámetro** el **nodo** que se va **a eliminar**. Además, esta función debe ser **invocada 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 HTML se invoca 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.

Acceso directo a los atributos HTML

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en **acceder y/o modificar sus atributos y propiedades**. Mediante DOM, es posible acceder de forma sencilla a todos los **atributos HTML** y todas las **propiedades CSS** de cualquier elemento de la página.

Los **atributos HTML** de los elementos de la página se transforman automáticamente en **propiedades de los nodos**. Para **acceder** a su valor, simplemente se indica el **nombre** del **atributo HTML detrás** del **nombre** del **nodo**.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el

enlace:

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

alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>
```

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función **document.getElementById()**. A continuación, se obtiene el **atributo href** del enlace mediante **enlace.href**. Para obtener por ejemplo el atributo **id**, se utilizaría **enlace.id**.

Las **propiedades CSS** no son tan fáciles de obtener como los atributos HTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el **atributo style**. El siguiente ejemplo obtiene el valor de la propiedad **margin** de la imagen:

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Aunque el funcionamiento es homogéneo entre **distintos navegadores**, los **resultados no** son **exactamente iguales.**

Si el **nombre** de una **propiedad CSS** es **compuesto**, se accede a su valor **modificando ligeramente su nombre**:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight); // muestra "bold"

<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en **eliminar todos los guiones medios** (-) y **escribir en mayúscula la letra siguiente** a cada guión medio. A continuación se muestran algunos ejemplos:

- **font-weight** se transforma en **fontWeight**
- **line-height** se transforma en **lineHeight**
- **border-top-style** se transforma en **borderTopStyle**
- **list-style-image** se transforma en **listStyleImage**

El único **atributo HTML** que **no tiene** el **mismo nombre** en **HTML** y en las **propiedades DOM** es el atributo **class**. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento HTML. En su lugar, **DOM utiliza el nombre className** para acceder al atributo **class** de HTML:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"

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