UD6
Utilización del modelo de objetos del documento (DOM)

1. DOM

La manipulación del **modelo de objetos del documento** -más conocido por sus siglas en inglés: DOM (Document Object Model) - es fundamental para el desarrollo de aplicaciones web, porque **sin esta capacidad no es posible alterar la visualización de las páginas dinámicamente**.

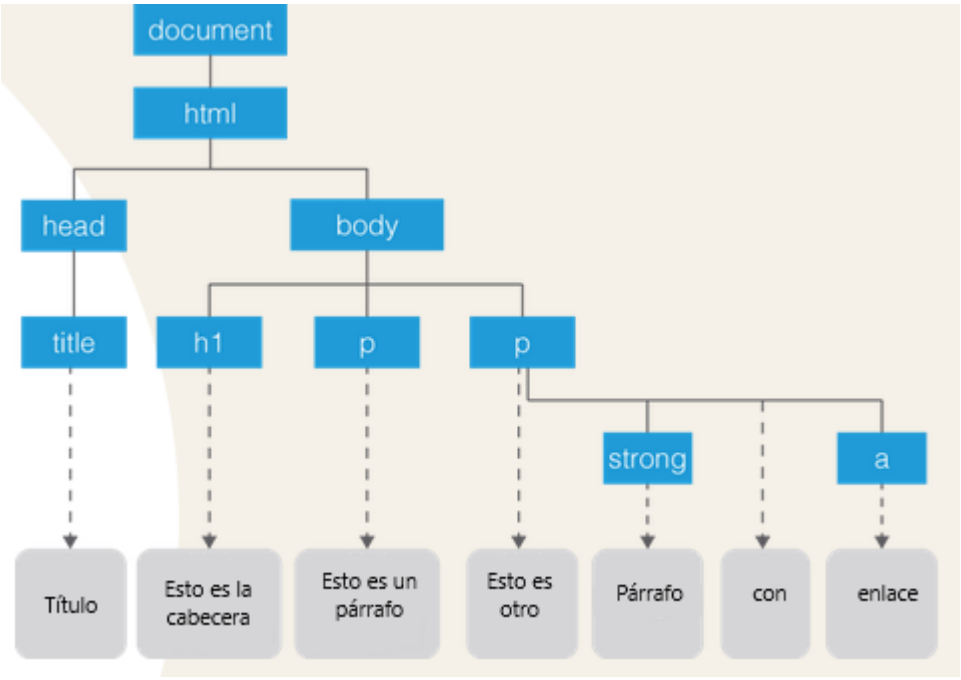
Internamente, los navegadores trabajan con los documentos web **como si se tratara de un árbol de nodos**, y es a partir de este árbol que se puede modificar la representación de la página, tanto **añadiendo nuevos elementos** (párrafos, cabeceras, tablas ...) como **modificando los atributos** de los nodos que ya se encuentran en el documento o **eliminándolos**.

Además, es posible **buscar tanto elementos concretos como listas de elementos** utilizando diferentes propiedades y métodos según sus necesidades: a través de relaciones (el primer elemento, el próximo elemento ...), buscando según el tipo de elemento, su identificador o haciendo una búsqueda más compleja gracias a los selectores de CSS.

1.1. Estructura del DOM

Veamos con un ejemplo cómo se desarrollan todas las **ramas** y **nodos** de un árbol DOM de las siguientes líneas de código:

```
<html>
   <head>
      <title>Título</title>
   </head>
   <body>
      <h1>Esto es la cabecera</h1>
         <p>Esto es un párrafo</p>
         <p>Esto es otro<strong>párrafo</strong> con <a>enlace</a></p>
   </body>
</html>
```



Como se puede apreciar, **la raíz del árbol es el nodo "document"** , aunque no forma parte del código HTML. A continuación encontramos el **nodo html** , que contiene los **nodos "head" y "body"** ; estos contienen otros nodos: **title , h1 y p** . **Todos estos nodos son de tipo Element** . En cambio, el último elemento de cada rama **(las hojas)** son de tipo Text y tienen una consideración diferente (TextNode).

Preste atención al último párrafo del documento: en este caso el texto incluye los elementos "strong" y "a" , pero estos elementos no cuelgan del texto sino que cuelgan del "nodo p" . Por tanto, **se puede concluir que un elemento siempre es descendiente de otro elemento y nunca de un texto**.

1.2. Obteniendo el texto de nodos

Existe principalmente 4 formas distintas de obtener texto de nodos HTML: nodeValue , innerHTML, textContent e innerText. Nosotros nos centraremos en el uso de:

- **innerHTML**: Obtiene el texto en formato **"text/html"** por lo que si el texto tuviese alguna etiqueta html la tendría en cuenta. Es un poco menos eficiente.
- **textContent** : Obtiene el texto en formato **"text/plain"**. Solo devuelve el texto tal cual por lo que su uso es un poco más eficiente que innerHTML.

Veamos un ejemplo:

```
<p id="primero"> Primer párrafo </p>
  <script>
      let primero=document.getElementById('primero');
      alert(primero.textContent);
      alert(primero.innerHTML);
  </script>
```

1.3. Relación entre nodos

Con estas propiedades de DOM podemos localizar, movernos por los distintos elementos del árbol de nodos generado.

1.3.1. parentNode

La propiedad "**parentNode"** permite acceder al padre del nodo o devuelve null si no existe (por ejemplo, si el nodo no se ha añadido al documento).

```
<body>
   <div id="contenedor">
      <p id="primero">Primer parrafo</p>
   </div>
<script>
var primero = document.getElementById('primero');
console.log(primero.parentNode.nodeName);                              //DIV
console.log("Padre del padre: " + primero.parentNode.parentNode.nodeName); //BODY
</script>
</body>
```

Nota: Como podemos ver también se puede concatenar y acceder al padre del padre.

1.3.2. firstElementChild y lastElementChild

Las propiedades **firstElementChild** y **lastElementChild** permiten acceder al primer y al último hijo de un nodo, como se puede comprobar en el ejemplo siguiente:

```
<div id="contenedor">
      <p id="primero">Primer párrafo</p>
      <p id="segundo">Segundo párrafo</p>
      <p id="tercero">Tercer párrafo</p></div>
<script>
      var contenedor = document.getElementById('contenedor');
      alert(contenedor.firstElementChild.textContent);  //Primer párrafo
      alert(contenedor.lastElementChild.textContent);   //Tercer párrafo
</script>
```

1.3.3. previousElementSibling y nextElementSibling

Por otra parte, **previousElementSibling** y **nextElementSibling** permiten acceder a los nodos hermanos, anterior y posterior respectivamente, de un mismo nodo. Sibling, en inglés, significa hermano/a. Ejemplo:

```
<div id="contenedor">
   <p id="primero">Primer párrafo</p>
   <p id="segundo">Segundo párrafo</p>
   <p id="tercero">Tercer párrafo</p></div>
<script>
   var contenedor = document.getElementById('segundo'); //Segundo párrafo
   alert(contenedor.textContent);
   alert(contenedor.previousElementSibling.textContent);   //Primer párrafo
   alert(contenedor.nextElementSibling.textContent);       //Tercer párrafo
</script>
```

1.3.4. Obtención de nodos descendientes: 'children' y 'hasChildNodes()'

La propiedad **children** nos permite acceder a la lista de nodos contenidos que se puede tratar como un array y conocer la cantidad de nodos a través de la propiedad **length** . Por otro lado, si sólo se quiere saber si contiene otros nodos o no, se puede invocar el método **hasChildNodes()** , que devolverá “true” si contiene o “false” en caso contrario.

```
<div id="contenedor">
      <p id="primero">Primer párrafo</p>
      <p id="segundo">Segundo párrafo</p>
      <p id="tercero">Tercer párrafo</p></div>
<script>
      var contenedor = document.getElementById('contenedor');
      console.log('El contenedor contiene nodos: ' + contenedor.hasChildNodes());
      console.log("Contiene " + contenedor.children.length + ' nodos');
      let hijo = contenedor.firstElementChild;
      while ( hijo != null)
      {
          console.log("El contenido del hijo es: "  + hijo.textContent );
          hijo = hijo.nextElementSibling;
```

```
      }
</script>
```

También podríamos iterar por los elementos como lo hemos visto en clase:

```
for( var i=0; i < contenedor.children.length; i++)
{
 console.log("-- El contenido del hijo es: " + contenedor.children[i].textContent );
}
```

1.4. Creación, sustitución y borrado de elementos

1.4.1. Creación de elementos

Para crear un nuevo elemento utilizaremos una serie de pasos con los que trabajaremos con una serie de métodos:

1. **createElement:** El método **document.createElement (<tipo_elemento>)** creará un nuevo elemento HTML del tipo que deseemos.
2. **createTextNode:** Para añadir contenido a dicho elemento recién creado debemos de utilizar el método **document.createTextNode (<contenido>)** el cual nos va a crear un "nodo" con el contenido deseado.
3. **appendChild:** Para añadir el nodo con el contenido al elemento HTML creado debemos de utilizar el método **elementoHTML.appendChild (<contenido>).** Este método es usado con propósito general para añadir elementos a nuestro árbol DOM. (También si quisieramos insertar el nodo por el principio se puede usar el método "**insertBefore()**").
4. Y por último también podríamos añadirle atributos a dicho elemento recién creado mediante el método **elementoHTML.setAttribute(<tipo_atributo>, <valor_atributo>).** (Este método se usa tanto para añadir como para modificar atributos que ya existen). Si quisiéramos eliminar atributos podríamos utilizar el método **elementoHTML.removeAttribute(<tipo_atributo>**).

Veamos un ejemplo de un método que crea un párrafo:

```
function crearParrafo() {
            //Crear elemento
            var parrafo = document.createElement("p");
            //Crear contenido
            var nodoTexto = document.createTextNode("Hola, soy un nuevo párrafo");
            parrafo.appendChild(nodoTexto);
            //Añadir atributos:
            parrafo.setAttribute("class", "miClase");
//parrafo.className="miClase";

    }
```

Nota: Debemos de fijarnos que en el anterior código hemos creado un nuevo párrafo, pero este no ha sido añadido al árbol DOM. Para añadirlo tendríamos que usar "appendChild" nuevamente.

Vamos a ver un nuevo ejemplo de cómo podríamos crear una imagen:

```
function crearImagen() {
	//Crear elemento
	var imagen = document.createElement("img");
	//Añadir atributos:
	//Le pedimos al usuario la ruta de la imagen
	var ruta = prompt("Introduce la ruta de la imagen");
	imagen.src = ruta;
	//Añadimos un nuevo atributo:
	imagen.setAttribute("alt", prompt("Introduce texto alternativo"));

	//var cont = document.getElementById("div1");   //Terminaríamos seleccionado
el
	//cont.appendChild(imagen);                     // elemento donde queremos
                                                    // añadir la imagen.
}
```

1.4.2. Borrado de elementos

Para borrar elementos necesitamos tener acceso a 2 elementos: el padre del elemento que queremos borrar y el elemento a borrar. Para esta operación utilizaremos el método **padre.removeChild(hijo).** Veámoslo con un ejemplo:

```
function borrarUltimoHijo() {
	var padre = document.getElementById("div1");
	var hijo = padre.lastChild; //Seleccionar el último hijo
	//Elimina un elemento hijo:
	padre.removeChild(hijo);
}
```

1.4.3. Sustitución de elementos

Para sustituir o reemplazar elementos necesitamos también 3 elementos: el nodo padre, el nodo hijo que queremos reemplazar, y el nuevo elemento por el que se va a reemplazar. Para este cometido utilizamos el método **padre.replaceChild (<nuevoElemento>, <elementoReemplazado>).** Veámoslo con un ejemplo:

```
function sustituirPrimero() {
	//Accedemos al padre:
	var padre = document.getElementById("div1");
	//Creamos el nuevo elemento: un párrafo con el texto Vacío
	var parrafo = document.createElement("p");
	var contenidoParrafo = document.createTextNode("Vacio");
	parrafo.appendChild(contenidoParrafo);
```

```
        //Sustituir un elemento:
        padre.replaceChild(parrafo, padre.firstChild);    //Nuevo, viejo
}
```

Ejercicio 1:

A partir de el siguiente código HTML:

```
<body>
        <div id="norte">
        <h2>Galicia</h2>
        <ul id="lista">
                <li>A Coruña</li>
                <li>Lugo</li>
                <li>Ourense</li>
                <li>Pontevedra</li>
        </ul>
        </div>
        <div id="sur">
        <h2>Canarias</h2>
        <ul id="lista">
                <li>Santa Cruz de Tenerife</li>
                <li>Las Palmas</li>
        </ul>
        </div>
        <div id="texto"></div>
</body>
```

deberá de recorrer los distintos nodos y obtener los valores de las provincias para que cuando el usuario haga "click" en la provincia Galicia deberá de aparecer como resultado un análisis de las provincias que tiene:

Galicia

- A Coruña
- Lugo
- Ourense
- Pontevedra

Canarias

- Santa Cruz de Tenerife
- Las Palmas

Has elegido Galicia que está situada en el norte
El número de provincias es 4: A Coruña Lugo Ourense Pontevedra

Lo mismo pasará cuando se hace click en las provincia "Canarias":

Galicia

- A Coruña
- Lugo
- Ourense
- Pontevedra

Canarias

- Santa Cruz de Tenerife
- Las Palmas

Has elegido Canarias que está situada en el sur
El número de provincias es 2: Santa Cruz de Tenerife Las Palmas

Aclaración para el ejercicio:
Debemos de tener un punto de partida para empezar a desgranar nuestro árbol HTML cuando necesitemos obtener la información requerida. Este punto de partida será el elemento clickeado (Galicia o Canarias) y podemos obtener ese nodo gracias al elemento **"e.target"** cuando se haga click sobre el mimo.

REFERENCIAS - FUENTES

Material Instituto Abierto de Cataluña:
https://ioc.xtec.cat/materials/FP/Materials/ICC0_DAW/DAW_ICC0_M06/web/html/index.html
https://javascript.info/

Javascript.info:
https://javascript.info/document