Páginas dinámicas

Cuando comenzamos en el mundo del **desarrollo web**, normalmente comenzamos por aprender a escribir etiquetado o **marcado HTML** y además, añadir **estilos CSS** para darle color, forma y algo de interacción. Sin embargo, a medida que avanzamos, nos damos cuenta que en cierta forma podemos estar bastante limitados.

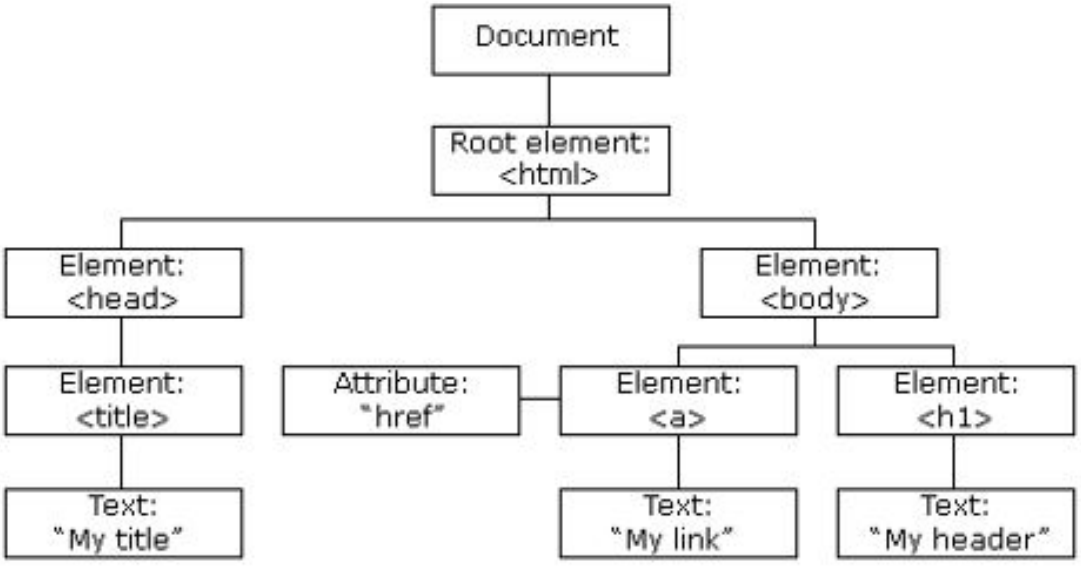
Si únicamente utilizamos HTML/CSS, sólo podremos crear **páginas «estáticas»** (*sin demasiada personalización por parte del usuario*), pero si añadimos Javascript, podremos crear **páginas «dinámicas»**. Cuando hablamos de páginas dinámicas, nos referimos a que podemos dotar de la **potencia** y **flexibilidad** que nos da un lenguaje de programación para crear documentos y páginas mucho más ricas, que brinden una experiencia más completa y con el que se puedan **automatizar** un gran abanico de tareas y **acciones**.

¿Qué es el DOM?

Las siglas DOM significan Document Object Model, o lo que es lo mismo, la **estructura** del documento **HTML**. Una página HTML está formada por múltiples **etiquetas** HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina árbol DOM (o simplemente DOM).

En **Javascript**, cuando nos referimos al **DOM** nos referimos a esta estructura, que podemos modificar de forma dinámica desde Javascript, añadiendo nuevas etiquetas, modificando o eliminando otras, cambiando sus atributos HTML, añadiendo clases, cambiando el contenido de texto, etc...

Al estar "amparado" por un **lenguaje de programación**, todas estas tareas se pueden automatizar, incluso indicando que se realicen cuando el usuario haga acciones determinadas, como por ejemplo: pulsar un botón, mover el mouse, hacer click en una parte del documento, escribir un texto, etc...



El objeto document

En Javascript, la forma de acceder al DOM es a través de un objeto llamado **document**, que representa el árbol DOM de la página o pestaña del navegador donde nos encontramos. En su interior pueden existir varios tipos de elementos, pero principalmente serán o :

- no es más que la representación genérica de una etiqueta: HTMLElement.
- es una unidad más básica, la cual puede ser o un nodo de texto.

Seleccionar elementos del DOM

Si nos encontramos en nuestro código Javascript y queremos hacer modificaciones en un elemento de la página HTML, lo primero que debemos hacer es buscar dicho elemento. Para ello, se suele intentar identificar el elemento a través de alguno de sus atributos más utilizados, generalmente el id o la clase.

Métodos tradicionales

Existen varios métodos, los más clásicos y tradicionales para realizar búsquedas de elementos en el documento. Observa que si lo que buscas es un elemento específico, lo mejor sería utilizar getElementById(), en caso contrario, si utilizamos uno de los 3 siguientes métodos, nos devolverá un donde tendremos que elegir el elemento en cuestión posteriormente:

Métodos de búsqueda	Descripción
ELEMENT `.getElementById(id)`	Busca el elemento HTML con el id **id**. Si no, devuelve NULL.
ARRAY `.getElementsByClassName(class)`	Busca elementos con la clase **class**. Si no, devuelve **[]**.
ARRAY `.getElementsByName(name)`	Busca elementos con atributo name **name**. Si no, devuelve **[]**.
ARRAY `.getElementsByTagName(tag)`	Busca elementos **tag**. Si no encuentra ninguno, devuelve **[]**.

Estos son los **4 métodos tradicionales** de Javascript para manipular el DOM. Se denominan tradicionales porque son los que existen en Javascript desde versiones más antiguas. Dichos métodos te permiten buscar elementos en la página dependiendo de los atributos **id**, **class**, **name** o de la propia etiqueta, respectivamen

getElementById()

El primer método, **.getElementById(id)** busca un elemento HTML con el **id** especificado en **id** por parámetro. En principio, un documento HTML bien construído **no debería** tener más de un elemento con el mismo **id**, por lo tanto, este método devolverá siempre un solo elemento:

```JS
const page = document.getElementById("page");   // <div id="page"></div>
```

getElementsByClassName()

Por otro lado, el método .getElementsByClassName(class) permite buscar los elementos con la clase especificada en class. Es importante darse cuenta del matiz de que el metodo tiene getElements en plural, y esto es porque al devolver clases (al contrario que los id) se pueden repetir, y por lo tanto, devolvernos varios elementos, no sólo uno.

```js
const items = document.getElementsByClassName("item");  // [div, div, div]

console.log(items[0]);      // Primer item encontrado: <div class="item"></div>
console.log(items.length);  // 3
```

> Estos métodos devuelven siempre un ARRAY con todos los elementos encontrados que encajen con el criterio. En el caso de no encontrar ninguno, devolverán un ARRAY vacío: `[]`.

Exactamente igual funcionan los métodos **getElementsByName(name)** y **getElementsByTagName(tag)**, salvo que se encargan de buscar elementos HTML por su atributo **name** o por su propia **etiqueta** de elemento HTML, respectivamente:

```js
// Obtiene todos los elementos con atributo name="nickname"
const nicknames = document.getElementsByName("nickname");

// Obtiene todos los elementos <div> de la página
const divs = document.getElementsByTagName("div");
```

Crear elementos en el DOM

Existen una serie de métodos para **crear de forma eficiente** diferentes elementos HTML o nodos, y que nos pueden convertir en una tarea muy sencilla el crear estructuras dinámicas, mediante bucles o estructuras definidas:

Métodos	Descripción
ELEMENT `.createElement(tag, options)`	Crea y devuelve el elemento HTML definido por el STRING **tag**.
NODE `.createComment(text)`	Crea y devuelve un nodo de comentarios HTML **<!-- text -->**.
NODE `.createTextNode(text)`	Crea y devuelve un nodo HTML con el texto **text**.
NODE `.cloneNode(deep)`	Clona el nodo HTML y devuelve una copia. **deep** es **false** por defecto.
BOOLEAN `.isConnected`	Indica si el nodo HTML está insertado en el documento HTML.

El método createElement()

Mediante el método **.createElement()** podemos crear un HTML **en memoria** (*¡no estará insertado aún en nuestro documento HTML!*). Con dicho elemento almacenado en una variable, podremos modificar sus características o contenido, para **posteriormente** insertarlo en una posición determinada del DOM o documento HTML.

Vamos a centrarnos en el proceso de **creación del elemento**, y en el próximo capítulo veremos el apartado de insertarlo en el DOM. El funcionamiento de **.createElement()** es muy sencillo: se trata de pasarle el nombre de la etiqueta **tag** a utilizar.

```js
const div = document.createElement("div");      // Creamos un <div></div>
const span = document.createElement("span");    // Creamos un <span></span>
const img = document.createElement("img");      // Creamos un <img>
```

Atributos HTML de un elemento

Hasta ahora, hemos visto cómo crear elementos HTML con Javascript, pero no hemos visto cómo modificar los atributos HTML de dichas etiquetas creadas. En general, una vez tenemos un elemento sobre el que vamos a crear algunos atributos, lo más sencillo es **asignarle valores como propiedades** de objetos:

```js
const div = document.createElement("div"); // <div></div>
div.id = "page";           // <div id="page"></div>
div.className = "data";    // <div id="page" class="data"></div>
div.style = "color: red"; // <div id="page" class="data" style="color: red"></div>
```

Eventos

Los scripts se dedican a esperar a que el usuario *"haga algo"* (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el mouse, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *"event handlers"* en inglés y suelen traducirse por *"manejadores de eventos"*.

Tipos de eventos

En este modelo, cada elemento o etiqueta HTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos HTML diferentes y un mismo elemento HTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pinchar y soltar el ratón	Todos los elementos
`ondblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`

`onkeydown`	Pulsar una tecla (sin soltar)	Elementos de formulario y `<body>`
`onkeypress`	Pulsar una tecla	Elementos de formulario y `<body>`
`onkeyup`	Soltar una tecla pulsada	Elementos de formulario y `<body>`
`onload`	La página se ha cargado completamente	`<body>`
`onmousedown`	Pulsar (sin soltar) un botón del ratón	Todos los elementos
`onmousemove`	Mover el ratón	Todos los elementos
`onmouseout`	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
`onmouseover`	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
`onmouseup`	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
`onreset`	Inicializar el formulario (borrar todos sus datos)	`<form>`
`onresize`	Se ha modificado el tamaño de la ventana del navegador	`<body>`
`onselect`	Seleccionar un texto	`<input>`, `<textarea>`
`onsubmit`	Enviar el formulario	`<form>`
`onunload`	Se abandona la página	`<body>`

	(por ejemplo al cerrar el navegador)	

Los eventos más utilizados en las aplicaciones web tradicionales son onload para esperar a que se cargue la página por completo, los eventos onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la *"acción por defecto"* de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo <input type="submit"> se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva.

Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *"manejador de eventos"* y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos HTML.
- Manejadores como funciones JavaScript externas.
- Manejadores *"semánticos"*.

Manejadores de eventos como atributos html

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento html. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias

por pinchar');" />
```

Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos html es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento html.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias
por pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}

<input type="button" value="Pinchame y verás"
onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento html se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable this y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {

  switch(elemento.style.borderColor) {

    case 'silver':

    case 'silver silver silver silver':

    case '#c0c0c0':

      elemento.style.borderColor = 'black';

      break;

    case 'black':

    case 'black black black black':

    case '#000000':

      elemento.style.borderColor = 'silver';

      break;

  }

}
```

```
<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">

  Sección de contenidos...

</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad `borderColor`.

Mezclar el código JavaScript con los elementos html solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos html para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás"
onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

```
// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento html
<input id="pinchable" type="button" value="Pinchame y verás" />
```