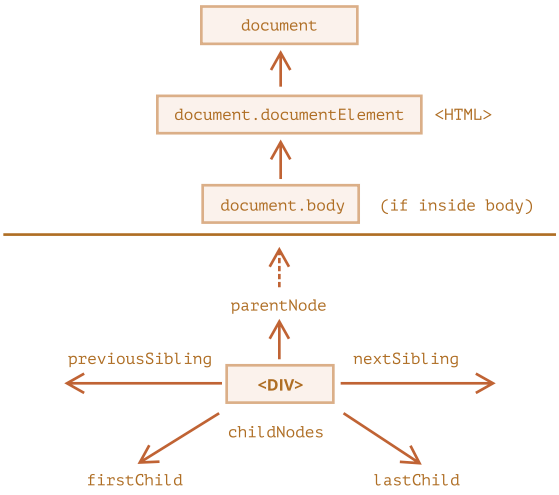
24 de octubre de 2022

# Recorriendo el DOM

El DOM nos permite hacer cualquier cosa con sus elementos y contenidos, pero lo primero que tenemos que hacer es llegar al objeto correspondiente del DOM.

Todas las operaciones en el DOM comienzan con el objeto `document`. Este es el principal "punto de entrada" al DOM. Desde ahí podremos acceder a cualquier nodo.

Esta imagen representa los enlaces que nos permiten viajar a través de los nodos del DOM:




Vamos a analizarlos con más detalle.

## En la parte superior: documentElement y body

Los tres nodos superiores están disponibles como propiedades de `document`:

**`<html>` = `document.documentElement`**

El nodo superior del documento es `document.documentElement`. Este es el nodo del DOM para la etiqueta `<html>`.

**`<body>` = `document.body`**

Otro nodo muy utilizado es el elemento `<body>` – `document.body`.

**`<head>` = `document.head`**

La etiqueta `<head>` está disponible como `document.head`.

**Hay una trampa: `document.body` puede ser `null`**

Un script no puede acceder a un elemento que no existe en el momento de su ejecución.

Por ejemplo, si un script está dentro de `<head>`, entonces `document.body` no está disponible, porque el navegador no lo ha leído aún.

Entonces, en el siguiente ejemplo `alert` muestra `null`:

```
<html>

<head>
  <script>
    alert( "From HEAD: " + document.body ); // null, no hay  <body> aún
  </script>
</head>

<body>

  <script>
    alert( "From BODY: " + document.body ); // HTMLBodyElement, ahora existe
  </script>

</body>
</html>
```

**En el mundo del DOM `null` significa "no existe"**

En el DOM, el valor `null` significa que "no existe" o "no hay tal nodo".

## Hijos: childNodes, firstChild, lastChild

Existen dos términos que vamos a utilizar de ahora en adelante:

- **Nodos hijos (childNodes)** – elementos que son hijos directos, es decir sus descendientes inmediatos. Por ejemplo, `<head>` y `<body>` son hijos del elemento `<html>`.
- **Descendientes** – todos los elementos anidados de un elemento dado, incluyendo los hijos, sus hijos y así sucesivamente.

Por ejemplo, aquí `<body>` tiene de hijos `<div>` y `<ul>` (y unos pocos nodos de texto en blanco):

```
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>
</body>
</html>
```

...Y los descendientes de `<body>` no son solo los hijos `<div>`, `<ul>` sino también elementos anidados más profundamente, como `<li>` (un hijo de `<ul>`) o `<b>` (un hijo de `<li>`) – el subárbol entero.

**La colección `childNodes` enumera todos los nodos hijos, incluidos los nodos de texto.**

El ejemplo inferior muestra todos los hijos de `document.body`:

```
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>Information</li>
  </ul>

  <div>End</div>

  <script>
    for (let i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i] ); // Texto, DIV, Texto, UL, ..., SCRIPT
    }
```

```
  </script>
  ...más cosas...
</body>
</html>
```

Por favor observa un interesante detalle aquí. Si ejecutamos el ejemplo anterior, el último elemento que se muestra es `<script>`. De hecho, el documento tiene más cosas debajo, pero en el momento de ejecución del script el navegador todavía no lo ha leído, por lo que el script no lo ve.

**Las propiedades `firstChild` y `lastChild` dan acceso rápido al primer y al último hijo.**

Son solo atajos. Si existieran nodos hijos, la respuesta siguiente sería siempre verdadera:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

También hay una función especial `elem.hasChildNodes()` para comprobar si hay algunos nodos hijos.

## Colecciones del DOM

Como podemos ver, `childNodes` parece un array. Pero realmente no es un array, sino más bien una *colección* – un objeto especial iterable, simil-array.

Hay dos importantes consecuencias de esto:

1. Podemos usar `for..of` para iterar sobre él:

```
for (let node of document.body.childNodes) {
  alert(node); // enseña todos los nodos de la colección
}
```

`Eso es porque es iterable (proporciona la propiedad `Symbol.iterator`, como se requiere).`

2. Los métodos de Array no funcionan, porque no es un array:

```
alert(document.body.childNodes.filter); // undefined (¡No hay método filter!)
```

La primera consecuencia es agradable. La segunda es tolerable, porque podemos usar `Array.from` para crear un array "real" desde la colección si es que queremos usar métodos del array:

```
alert( Array.from(document.body.childNodes).filter ); // función
```

### Las colecciones DOM son solo de lectura

Las colecciones DOM, incluso más-- *todas* las propiedades de navegación enumeradas en este capítulo son sólo de lectura.

No podemos reemplazar a un hijo por otro elemento asignándolo así `childNodes[i] = ...`.

Cambiar el DOM necesita otros métodos. Los veremos en el siguiente capítulo.

### Las colecciones del DOM están vivas

Casi todas las colecciones del DOM, salvo algunas excepciones, están *vivas*. En otras palabras, reflejan el estado actual del DOM.

Si mantenemos una referencia a `elem.childNodes`, y añadimos o quitamos nodos del DOM, entonces estos nodos aparecen en la colección automáticamente.

**No uses `for..in` para recorrer colecciones**

Las colecciones son iterables usando `for..of`. Algunas veces las personas tratan de utilizar `for..in` para eso.

Por favor, no lo hagas. El bucle `for..in` itera sobre todas las propiedades enumerables. Y las colecciones tienen unas propiedades "extra" raramente usadas que normalmente no queremos obtener:

```
<body>
<script>
  // enseña 0, 1, longitud, item, valores y más cosas.
  for (let prop in document.body.childNodes) alert(prop);
</script>
</body>
```

## Hermanos y el padre

*Los hermanos* son nodos que son hijos del mismo padre.

Por ejemplo, aquí `<head>` y `<body>` son hermanos:

```
<html>
  <head>...</head><body>...</body>
</html>
```

- `<body>` se dice que es el hermano "siguiente" o a la "derecha" de `<head>`,
- `<head>` se dice que es el hermano "anterior" o a la "izquierda" de `<body>`.

El hermano siguiente está en la propiedad `nextSibling` y el anterior – en `previousSibling`.

El padre está disponible en `parentNode`.

Por ejemplo:

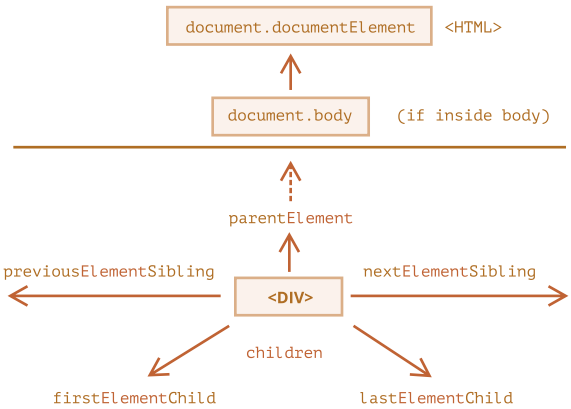
```
// el padre de <body> es <html>
alert( document.body.parentNode === document.documentElement ); // verdadero

// después de <head> va <body>
alert( document.head.nextSibling ); // HTMLBodyElement

// antes de <body> va <head>
alert( document.body.previousSibling ); // HTMLHeadElement
```

## Navegación solo por elementos

Las propiedades de navegación enumeradas abajo se refieren a *todos* los nodos. Por ejemplo, en `childNodes` podemos ver nodos de texto, nodos elementos; y si existen, incluso los nodos de comentarios.

Pero para muchas tareas no queremos los nodos de texto o comentarios. Queremos manipular el nodo que representa las etiquetas y formularios de la estructura de la página.

Así que vamos a ver más enlaces de navegación que solo tienen en cuenta los *elementos nodos*:




Los enlaces son similares a los de arriba, solo que tienen dentro la palabra `Element`:

- `children` – solo esos hijos que tienen el elemento nodo.
- `firstElementChild`, `lastElementChild` – el primer y el último elemento hijo.

- `previousElementSibling`, `nextElementSibling` – elementos vecinos.
- `parentElement` – elemento padre.

> **¿Por qué `parentElement`? ¿Puede el padre *no* ser un elemento?**
>
> La propiedad `parentElement` devuelve el "elemento" padre, mientras `parentNode` devuelve "cualquier nodo" padre. Estas propiedades son normalmente las mismas: ambas seleccionan el padre.
>
> Con la excepción de `document.documentElement`:
>
> ```
> 1  alert( document.documentElement.parentNode ); // documento
> 2  alert( document.documentElement.parentElement ); // null
> ```
>
> La razón es que el nodo raíz `document.documentElement` (`<html>`) tiene a `document` como su padre. Pero `document` no es un elemento nodo, por lo que `parentNode` lo devuelve y `parentElement` no lo hace.
>
> Este detalle puede ser útil cuando queramos navegar hacia arriba desde cualquier elemento `elem` al `<html>`, pero no hacia el `document`:
>
> ```
> 1  while(elem = elem.parentElement) { // sube hasta <html>
> 2    alert( elem );
> 3  }
> ```

Vamos a modificar uno de los ejemplos de arriba: reemplaza `childNodes` por `children`. Ahora enseña solo elementos:

```
1  <html>
2  <body>
3    <div>Begin</div>
4
5    <ul>
6      <li>Information</li>
7    </ul>
8
9    <div>End</div>
10
11   <script>
12     for (let elem of document.body.children) {
13       alert(elem); // DIV, UL, DIV, SCRIPT
14     }
15   </script>
16   ...
17 </body>
18 </html>
```

## Más enlaces: tablas

Hasta ahora hemos descrito las propiedades de navegación básicas.

Ciertos tipos de elementos del DOM pueden tener propiedades adicionales, específicas de su tipo, por conveniencia.

Las tablas son un gran ejemplo de ello, y representan un particular caso importante:

**El elemento `<table>`** soporta estas propiedades (añadidas a las que hemos dado anteriormente):

- `table.rows` – la colección de elementos `<tr>` de la tabla.
- `table.caption/tHead/tFoot` – referencias a los elementos `<caption>`, `<thead>`, `<tfoot>`.
- `table.tBodies` – la colección de elementos `<tbody>` (pueden ser muchos según el estándar, pero siempre habrá al menos uno, aunque no esté en el HTML el navegador lo pondrá en el DOM).

**`<thead>`, `<tfoot>`, `<tbody>`** estos elementos proporcionan las propiedades de las `filas`.

- `tbody.rows` – la colección dentro de `<tr>`.

**`<tr>`:**

- `tr.cells` – la colección de celdas `<td>` y `<th>` dentro del `<tr>` dado.
- `tr.sectionRowIndex` – la posición (índice) del `<tr>` dado dentro del `<thead>/<tbody>/<tfoot>` adjunto.
- `tr.rowIndex` – el número de `<tr>` en la tabla en su conjunto (incluyendo todas las filas de una tabla).

**`<td>` and `<th>`:**

- `td.cellIndex` – el número de celdas dentro del adjunto `<tr>`.

Un ejemplo de uso:

```
<table id="table">
  <tr>
    <td>one</td><td>two</td>
  </tr>
  <tr>
    <td>three</td><td>four</td>
  </tr>
</table>

<script>
  // seleccionar td con "dos" (primera fila, segunda columna)
  let td = table.rows[0].cells[1];
  td.style.backgroundColor = "red"; // destacarlo
</script>
```

La especificación: tabular data.

También hay propiedades de navegación adicionales para los formularios HTML. Las veremos más adelante cuando empecemos a trabajar con los formularios.

## Resumen

Dado un nodo del DOM, podemos ir a sus inmediatos vecinos utilizando las propiedades de navegación.

Hay dos conjuntos principales de ellas:

- Para todos los nodos: `parentNode`, `childNodes`, `firstChild`, `lastChild`, `previousSibling`, `nextSibling`.
- Para los nodos elementos: `parentElement`, `children`, `firstElementChild`, `lastElementChild`, `previousElementSibling`, `nextElementSibling`.

Algunos tipos de elementos del DOM, por ejemplo las tablas, proveen propiedades adicionales y colecciones para acceder a su contenido.

## Tareas

### DOM children

importancia: 5

Mira esta página:

```
<html>
<body>
  <div>Users:</div>
  <ul>
    <li>John</li>
    <li>Pete</li>
  </ul>
</body>
</html>
```

Para cada una de las siguientes preguntas, da al menos una forma de cómo acceder a ellos:

- ¿El nodo `<div>` del DOM?
- ¿El nodo `<ul>` del DOM?
- El segundo `<li>` (con Pete)?

solución

### La pregunta de los hermanos

importancia: 5

Si `elem` – es un elemento nodo arbitrario del DOM...

- ¿Es cierto que `elem.lastChild.nextSibling` siempre es `null`?
- ¿Es cierto que `elem.children[0].previousSibling` siempre es `null`?

solución

## Seleccionar todas las celdas diagonales

importancia: 5

Escribe el código para pintar todas las celdas diagonales de rojo.

Necesitarás obtener todas las `<td>` de la `<table>` y pintarlas usando el código:

```
// td debe ser la referencia a la celda de la tabla
td.style.backgroundColor = 'red';
```

El resultado debe ser:

| | | | | |
|---|---|---|---|---|
| 1:1 | 2:1 | 3:1 | 4:1 | 5:1 |
| 1:2 | 2:2 | 3:2 | 4:2 | 5:2 |
| 1:3 | 2:3 | 3:3 | 4:3 | 5:3 |
| 1:4 | 2:4 | 3:4 | 4:4 | 5:4 |
| 1:5 | 2:5 | 3:5 | 4:5 | 5:5 |

Abrir un entorno controlado para la tarea.

solución

Lección anterior Próxima lección

Compartir

Mapa del Tutorial

## Comentarios

- Si tiene sugerencias sobre qué mejorar, por favor enviar una propuesta de GitHub o una solicitud de extracción en lugar de comentar.
- Si no puede entender algo en el artículo, por favor explique.
- Para insertar algunas palabras de código, use la etiqueta `<code>`, para varias líneas – envolverlas en la etiqueta `<pre>`, para más de 10 líneas – utilice una entorno controlado (sandbox) (plnkr, jsbin, codepen...)

1 Comentario

Acceder ▼

Únete a la conversación...

INICIAR SESIÓN CON

O REGISTRARSE CON DISQUS

Nombre

10 Comparte

Mejores Más recientes Más antiguos

Gustavo Redondo

hace un año

Vista – uploads.disquscdn.com

0 0 Responder Comparte ›

Suscríbete Política de Privacidad No vendan mis datos