Navegando no DOM e a Relação Entre Elementos em JavaScript

1. A ESTRUTURA FAMILIAR DOS ELEMENTOS HTML

Para interagir de forma eficaz com uma página da web usando JavaScript, é essencial entender o ***Document Object Model*** **(DOM)**. No entanto, em vez de enxergá-lo como uma simples lista de elementos, devemos **percebê-lo como uma árvore hierárquica**. Nessa estrutura, cada elemento possui uma relação com os outros, semelhante a uma árvore genealógica. Compreender essas relações de "parentesco" — quem é o pai, o filho ou o irmão de um elemento — é fundamental para selecionar e manipular exatamente o que desejamos de forma eficiente e precisa. Para facilitar esse entendimento, vamos explorar três conceitos centrais de relacionamento que formam a base da navegação no DOM:

- **Elemento Pai (Parent):** O elemento que contém outros elementos diretamente dentro dele. Na nossa analogia, seria o ancestral direto.
- **Elementos Filhos (Children):** Os elementos que estão diretamente contidos dentro de um elemento pai. Eles são os descendentes diretos.
- **Elementos Irmãos (Siblings):** Elementos que compartilham o mesmo pai e, portanto, estão no mesmo nível hierárquico. O termo `siblings` é usado porque, diferente de termos mais específicos como `brothers` ou `sisters` em inglês, ele é neutro e engloba todos os elementos no mesmo nível, o que reflete com precisão o funcionamento do DOM.

Com esses conceitos em mente, vamos aprofundar em como o JavaScript os interpreta e como podemos utilizar essas relações para navegar pela estrutura da nossa página.

1.1. A Diferença de Nodes vs. Elements

Um ponto que frequentemente causa confusão para desenvolvedores iniciantes é a diferença entre "Nós" (*Nodes*) e "Elementos" (Elements). Para o navegador, a árvore do DOM é composta por Nós. Um nó pode ser uma *tag* HTML (um elemento), mas também pode ser um texto, um comentário ou até mesmo os espaços em branco e quebras de linha que inserimos no nosso código HTML para organizá-lo.

Essa distinção é crucial. O instrutor na fonte demonstra isso perfeitamente nas ferramentas de desenvolvedor do navegador. Ao inspecionar uma `div` que continha seis `div`s filhas, a propriedade `childNodes` retornou uma contagem de **13**, enquanto a propriedade `children` retornou **6**. Essa

discrepância é causada pelo fato de `childNodes` contar cada nó de texto — incluindo os espaços e quebras de linha entre as *tags* `div` — enquanto `children` isola corretamente apenas os elementos HTML que normalmente queremos manipular.

A tabela abaixo contrasta as propriedades de navegação focadas em **Nós** com suas contrapartes focadas em **Elementos**, que são geralmente as que você irá preferir usar. Você pode verificar isso por conta própria inspecionando um elemento nas **Ferramentas de Desenvolvedor** do seu navegador e observando o painel "Properties".

Propriedade focada em **Nodes**	Propriedade focada em **Elements**	**Descrição da Diferença**
`childNodes`	`children`	`childNodes` retorna todos os nós filhos, incluindo textos e comentários. `children` retorna apenas os nós que são elementos HTML.
`firstChild`	`firstElementChild`	`firstChild` retorna o primeiro nó filho. Isso é crucial porque o `firstChild` é frequentemente um nó de texto indesejado (um espaço ou quebra de linha) deixado no HTML. `firstElementChild` retorna o primeiro elemento HTML filho.
`lastChild`	`lastElementChild`	`lastChild` retorna o último nó filho, que também pode ser um texto. `lastElementChild` retorna o último elemento HTML filho.
`nextSibling`	`nextElementSibling`	`nextSibling` aponta para o próximo nó irmão. Usá-lo pode retornar inesperadamente um nó de texto, quebrando a lógica que espera outro elemento HTML. `nextElementSibling` move-se de forma confiável entre os elementos reais.
`parentNode`	`parentElement`	Ambos geralmente apontam para o elemento pai, sendo usados para "subir" na árvore do DOM. `parentElement` é a opção mais comum e segura para garantir que você obtenha um elemento HTML.

Com essa distinção teórica clara, vamos ver como aplicar essas propriedades em código para navegar pelo DOM de forma eficaz.

1.2. Acessando Elementos na Prática com JavaScript

Agora que entendemos a teoria por trás das relações do DOM e a diferença entre nós e elementos, vamos aplicar esse conhecimento em exemplos práticos. Os trechos de código a seguir demonstram como usar as propriedades de relacionamento para navegar pela árvore DOM a partir de um elemento de referência.

1.2.1. Acessando Elementos Filhos (`Children`)

A forma mais comum de obter os descendentes diretos de um elemento é através da propriedade `.children`. Ela retorna uma `HTMLCollection`, que é uma coleção semelhante a um *array* contendo todos os elementos HTML filhos de um determinado pai.

```
// Supondo que 'caixa1' seja a nossa div principal que contém outros elementos
const caixa1 = document.querySelector("#caixa1");

// Acessando todos os elementos filhos
console.log(caixa1.children);
```

O resultado deste código será uma coleção de todos os elementos `<div>` que estão diretamente dentro de `#caixa1`. Como se trata de uma coleção, podemos acessar elementos individuais através de seu índice. Por exemplo, usar `caixa1.children[4]` selecionaria o quinto elemento filho, que no exemplo do instrutor era a `div` contendo o texto "React".

1.2.2. Selecionando o Primeiro e o Último Filho

Para selecionar especificamente o primeiro ou o último elemento filho, JavaScript oferece atalhos convenientes e muito legíveis: as propriedades `.firstElementChild` e `.lastElementChild`.

```
// Pegando o primeiro elemento filho
console.log(caixa1.firstElementChild);

// Pegando o último elemento filho
console.log(caixa1.lastElementChild);
```

Além do atalho `.lastElementChild`, também é possível obter o último filho de forma dinâmica usando a propriedade `.length` da coleção `children`. Lembre-se que, como os índices de *arrays* começam em `0`, o índice do último elemento é sempre o comprimento total da coleção menos um.

```
// Método alternativo para pegar o último elemento
const todosOsFilhos = caixa1.children;
console.log(todosOsFilhos[todosOsFilhos.length - 1]);
```

1.2.3. Identificando o Elemento Raiz do Documento

Todo documento HTML possui um "nó raiz" (*root node*), que serve como o ponto de partida de toda a estrutura. Em um contexto de desenvolvimento web padrão, esse nó raiz é o próprio objeto `document`. Podemos confirmar isso a partir de qualquer elemento na página.

O método `.getRootNode()` e a propriedade `.ownerDocument` nos permitem identificar o proprietário do documento de um determinado elemento. Note a diferença de sintaxe: `.getRootNode()` é um **método**, por isso requer parênteses `()` para ser executado, enquanto `.ownerDocument` é uma **propriedade**, que é acessada diretamente sem parênteses. Essa distinção entre métodos e propriedades é um conceito central em JavaScript.

```
// Obtendo o nó raiz a partir de um elemento
// 'c1' seria um dos cursos dentro da caixa1
const c1 = document.querySelector(".curso");

console.log(c1.getRootNode());
console.log(c1.ownerDocument);
```

No navegador, para uma página web padrão, ambos os comandos acima retornarão o objeto `document`, confirmando que ele é a raiz de toda a estrutura com a qual estamos interagindo.

2. CONCLUSÃO

Navegar pelo DOM com base nas relações entre elementos é uma habilidade poderosa e fundamental em JavaScript. Ao dominar esses conceitos, você ganha precisão e flexibilidade para encontrar e manipular qualquer elemento na página. Vamos recapitular os pontos-chave que abordamos:

- **A importância de visualizar o DOM como uma árvore de relações:** Entender os conceitos de pai (`parent`), filho (`child`) e irmão (`sibling`) é o primeiro passo para uma manipulação eficaz.
- **A diferença fundamental entre "Nodes" e "Elements":** Lembre-se que *Nodes* incluem textos e comentários, enquanto *Elements* se referem especificamente às tags HTML. Dê preferência às propriedades focadas em elementos (`.children`, `.firstElementChild`, etc.) para evitar resultados inesperados.
- **As principais propriedades JavaScript para navegação:** Vimos como usar `.children` para obter uma coleção de filhos, `.parentElement` para subir na hierarquia e atalhos como `.firstElementChild` e `.lastElementChild` para uma seleção direta.

Este conhecimento serve como uma base sólida para manipulações mais complexas do DOM. Dominar essas relações é um passo fundamental, então continue praticando, pois este tópico é de importância crítica para construir aplicações web dinâmicas.