Javascript: Navegação pela DOM

Sumário

- **Introdução**
  - O que é a DOM?
  - Por que navegar pela DOM é importante em JavaScript?
  - Navegação pelo DOM
- **Propriedades para navegação**
  - parentNode
  - childNodes
  - firstChild
  - LastChild
  - nextSibling
  - previousSibling

Sumário

- **Métodos de Seleção avançada**
  - element.closest()
  - element.querySelectorAll()

01

Introdução

Capítulo 1: Introdução

O que é a DOM?

A DOM (Document Object Model) é uma interface de programação para documentos HTML e XML. Ela representa a estrutura de um documento como uma árvore de nós, onde cada nó corresponde a uma parte do documento, como elementos, atributos e textos. A DOM permite que os desenvolvedores acessem, manipulem e atualizem o conteúdo, a estrutura e o estilo de um documento de forma dinâmica. Em outras palavras, a DOM é a ponte entre os documentos web e os scripts que interagem com eles, como o JavaScript.

Por que navegar pela DOM é importante em JavaScript?

Navegar pela DOM é crucial em JavaScript porque permite aos desenvolvedores criar interações dinâmicas e responsivas nos sites. Através da navegação pelo DOM, é possível:

1. Modificar o Conteúdo: Adicionar, remover ou alterar elementos e textos dentro do documento.

Capítulo 1: Introdução

3. Interagir com o Usuário: Responder a eventos do usuário, como cliques, digitação e movimentos do mouse, para criar uma experiência interativa.
4. Atualizar Estilos: Modificar dinamicamente os estilos CSS dos elementos para melhorar a apresentação visual.
5. Gerenciar Estrutura: Reorganizar elementos dentro da árvore do documento para refletir mudanças no layout ou na estrutura de dados.

Navegação pela DOM

Ao navegar pela DOM, os desenvolvedores podem criar aplicativos web mais ricos e funcionais, oferecendo uma experiência de usuário aprimorada. Além disso, a manipulação da DOM é uma habilidade fundamental para qualquer desenvolvedor JavaScript, pois é a base para muitas bibliotecas e frameworks populares, como React, Angular e Vue.

Exemplo de DOM na próxima figura.

Capítulo 1: Introdução

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Tarefas</title>
    <style> {Estilizacao aqui...} </style>
</head>
<body>
    <div class="container">
        <h1>Lista de Tarefas</h1>
        <input type="text" id="taskInput" placeholder="Nova tarefa...">
        <button onclick="addTask()">Adicionar</button>
        <ul id="taskList"></ul>
    </div>

    <script>
        function addTask() {
            const taskInput = document.getElementById('taskInput');
            const taskList = document.getElementById('taskList');

            // Cria um novo elemento de lista
            const li = document.createElement('li');
            li.textContent = taskInput.value;

            // Cria um botão de remover
            const removeButton = document.createElement('button');
            removeButton.textContent = 'Remover';
            removeButton.onclick = function() {
                taskList.removeChild(li);
            };

            // Adiciona o botão ao elemento de lista
            li.appendChild(removeButton);

            // Adiciona o elemento de lista à lista de tarefas
            taskList.appendChild(li);

            // Limpa o campo de entrada
            taskInput.value = '';
        }
    </script>
</body>
</html>
```

02

Propriedades para navegação

Propriedades para navegação

parentNode: Retorna o nó pai do elemento atual.

```
const child = document.querySelector('.child');
const parent = child.parentNode;
console.log(parent);
```

- Neste exemplo, parentNode retorna o elemento pai do nó child.

childNodes: Retorna uma coleção de todos os nós filhos do elemento, incluindo nós de texto e comentários.

```
const parent = document.querySelector('.parent');
const children = parent.childNodes;
console.log(children);
```

- Aqui, childNodes retorna todos os nós filhos do elemento parent.

Propriedades para navegação

firstChild: Retorna o primeiro nó filho do elemento.

```
const parent = document.querySelector('.parent');
const firstChild = parent.firstChild;
console.log(firstChild);
```

- firstChild retorna o primeiro nó filho do elemento parent.

lastChild: Retorna o último nó filho do elemento.

```
const parent = document.querySelector('.parent');
const lastChild = parent.lastChild;
console.log(lastChild);
```

- lastChild retorna o último nó filho do elemento parent.

Propriedades para navegação

nextSibling: Retorna o próximo nó irmão do elemento.

```
const firstChild = document.querySelector('.child');
const nextSibling = firstChild.nextSibling;
console.log(nextSibling);
```

- nextSibling retorna o próximo nó irmão do firstChild.

previousSibling: Retorna o nó irmão anterior do elemento.

```
const lastChild = document.querySelector('.child:last-child');
const previousSibling = lastChild.previousSibling;
console.log(previousSibling);
```

- previousSibling retorna o nó irmão anterior do lastChild.

03

Métodos de Seleção Avançada

Métodos de Seleção Avançada

O que são métodos de seleção avançada

Os métodos de seleção avançada no DOM são ferramentas poderosas que permitem localizar e manipular elementos específicos com precisão usando seletores CSS. Esses métodos oferecem uma maneira eficiente de navegar e interagir com a estrutura do documento, tornando o desenvolvimento de funcionalidades complexas mais fácil e intuitivo. Vamos explorar dois desses métodos:

1- element.closest(selector):

Este método retorna o ascendente mais próximo do elemento atual que corresponda ao seletor CSS fornecido. É especialmente útil para encontrar um elemento pai específico em uma estrutura de aninhamento complexa. Por exemplo, ao clicar em um botão dentro de um formulário, você pode usar closest para encontrar o formulário mais próximo.

Métodos de Seleção Avançada

```
const subItem = document.querySelector('#sub-item1');
const closestItem = subItem.closest('.item');
console.log(closestItem);
```

- No exemplo acima, closest encontra o elemento pai mais próximo de #sub-item1 que tenha a classe item.

2- element.querySelectorAll(selector)

```
const allItems = document.querySelectorAll('.container .item');
allItems.forEach(item => console.log('Item:', item));
```

- No exemplo acima, closest encontra o elemento pai mais próximo de #sub-item1 que tenha a classe item.

Métodos de Seleção Avançada

```
const allItems = document.querySelectorAll('.container .item');
allItems.forEach(item => console.log('Item:', item));
```

- No exemplo, querySelectorAll seleciona todos os elementos com a classe item dentro do contêiner e itera sobre eles para realizar ações específicas.

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, e diagramado por humano.

O conteúdo gerado foi por fins didádicos de construção, não realizado uma validação cuidadosamente humana no conteúdo e pode conter erros gerados por uma IA

Contatos: *Instagram* - *Github* - *LinkedIn*