Unidade VII: DOM soulcode.com

**1 DOM**

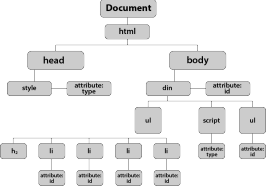
O DOM (Document Object Model) é uma das maiores funcionalidades da lin guagem Javascript, com ela poderemos manipular dinamicamente as páginas que construirmos, tornando os sites mais intuitivos e atrativos.

**1.1 O que é o DOM?**

É um recurso de programação para páginas HTML para que possamos mani pular o seu conteúdo dinamicamente. Ele fornece uma representação estru turada do documento como uma árvore.

O DOM define métodos que permitem acesso à árvore, para que eles pos sam alterar a estrutura, estilo e conteúdo do documento.

Figura 1: Estrutura de árvore

Fonte: josephchavez on Medium.

Cada página gera seu próprio DOM e este fica armazenado na memória do navegador. Podemos observar na Figura 1, uma hierarquia de elementos começando das tags superiores **html**, **head** e **body**.

Chamamos cada elemento do HTML nesta hierarquia de **nó**, com os recur sos do DOM poderemos manipular estes nós em quesito de estilos e compor tamento.

Observe que alguns nós possuem outros elementos dentro, estes são os descendentes de um nó, como por exemplo, o elemento <ul> que possui os descendentes h2 e li.

Na realidade a DOM detalha todas as propriedades e métodos destes ele mentos como objetos conforme estudamos anteriormente. Dessa forma, po

1

Unidade VII: DOM soulcode.com

demos unir nossa lógica de programação aos diversos recursos da DOM para gerar efeitos interessantes.

**1.2 Acessando elementos do documento**

**1.2.1 Por ID**

Podemos obter um elemento específico usando o ID atribuído a ele. Para isso, usamos o método ***getElementById***.

Listagem 1: getElementById

1 <div id="meu-elemento">Este eh um elemento</div>

2 ...

3 const elemento = document.getElementById("meu-elemento");

4 console.log(elemento); // Retorna o elemento com o ID "meu-elemento"

Uma busca é realizada no documento, em busca de um elemento que pos sua o ID passado como argumento, caso não exista o retorno da função será *undefined*.

**1.2.2 Por seletor CSS**

Podemos selecionar elementos usando seletores CSS com o método ***query Selector*** ou ***querySelectorAll***. O ***querySelector*** retorna o primeiro elemento correspondente ao seletor especificado, enquanto o ***querySelectorAll*** retorna uma lista de todos os elementos correspondentes.

Listagem 2: querySelector e querySelectorAll

1 <ul>

2 <li class="item">Item 1</li>

3 <li class="item">Item 2</li>

4 <li class="item">Item 3</li>

5 </ul>

6 ...

7 const primeiroItem = document.querySelector('.item');

8 console.log(primeiroItem); // Retorna o primeiro elemento com a classe "item" 9

10 const todosItens = document.querySelectorAll('.item');

11 console.log(todosItens); // Retorna uma NodeList com todos os elementos com a classe "item"

O argumento passado para ***querySelector/querySelectorAll*** deve ser um se letor CSS, isso se aplica a quaisquer um dos seletores possíveis: #id, classe, tipo, pseudo-elementos e etc.

**1.2.3 Outras formas**

Existem diversas outras formas de buscar vários elementos na página, vale ressaltar:

• ***getElementsByTagName***: Busca elementos a partir do nome da tag; 2

Unidade VII: DOM soulcode.com

• ***getElementsByClassName***: Busca elementos a partir do nome da classe;

• ***getElementsByName***: Busca elementos a partir do atributo name, co mum em inputs.

**1.3 Manipulando conteúdos e estilos**

Podemos modificar o conteúdo dos elementos HTML, como texto e atributos, utilizando propriedades específicas do DOM.

**1.3.1 textContent**

Esta propriedade é utilizada para modificar o texto interno do elemento, vale ressaltar que esta propriedade ignora o uso das tags.

Listagem 3: textContent

1 <div id="meu-elemento">Este eh um elemento</div>

2 ...

3 const elemento = document.getElementById('meu-elemento');

4 elemento.textContent = '<b>Novo texto do elemento</b>';

**1.3.2 innerHTML**

Esta propriedade é mais genérica que ***textContent***, pois ela pode ser usada para mudar as tags exibidas dentro de um elemento.

Listagem 4: innerHTML

1 <div id="meu-elemento">Este eh um elemento</div>

2 ...

3 const elemento = document.getElementById('meu-elemento');

4 elemento.innerHTML += '<b>Novo texto do elemento</b>';

5 elemento.innerHTML += '<b>Outro elemento</b>';

6 elemento.innerHTML += '<b>Mais um elemento</b>';

Dessa forma, veremos as tags <b> realmente serem interpretadas pelo nave gador. O uso do innerHTML deve ser feito com cuidado para evitar que alguns elementos já existentes não sejam removidos.

No exemplo da listagem 4, isso não acontece, pois o conteúdo textual ori ginal permanece dentro da div, ocorrendo apenas a concatenação com as tags <b>.

**1.3.3 src**

Podemos alterar dinamicamente o atributo ***src*** das imagens, dentre outros.

Listagem 5: Alterando atributo src

1 <img id="minha-imagem" src="imagem.jpg">

2 ...

3 const imagem = document.getElementById('minha-imagem');

4 imagem.src = 'nova-imagem.jpg';

3

Unidade VII: DOM soulcode.com

**1.3.4 style**

Podemos alterar os estilos dos elementos HTML utilizando as propriedades de estilo disponíveis no DOM.

Listagem 6: Estilização via DOM

1 <div id="meu-elemento">Este eh um elemento</div>

2 ...

3 const elemento = document.getElementById('meu-elemento');

4 elemento.style.backgroundColor = 'red';

5 elemento.style.color = 'black';

Observe, que esta ação do código acima irá adicionar o atributo style dentro do elemento. Podemos usar este atributo para alterar qualquer estilo exis tente.

É importante ressaltar a notação usada para as propriedades CSS, como estamos lidando com Javascript, o uso do hífen, comumente adotado no CSS não pode ser utilizado.

Dessa forma, *background-color* vira **backgroundColor**, *font-size* vira **font Size** e assim por diante, adotando o camel case.

**1.3.5 class**

Podemos adicionar ou remover classes dinamicamente fazendo uso do objeto ***classList***.

Listagem 7: classList

1 <div id="meu-elemento" class="classe1">Este eh um elemento</div> 2 ...

3 const elemento = document.getElementById('meu-elemento');

4 elemento.classList.add('classe2'); // Adiciona a classe "classe2" 5 elemento.classList.remove('classe1'); // Remove a classe "classe1" 6 elemento.classList.toggle('batata'); // Alterna a classe "batata"

Com estes métodos podemos adicionar, remover, substituir e alternas classes já definidas na folha de estilos do CSS.

**1.4 Manipulando eventos**

Os eventos permitem que o código JavaScript responda a ações do usuário, como cliques de mouse, pressionamentos de tecla, movimentos do mouse, entre outros.

**1.4.1 Atributo HTML**

É possível definir eventos diretamente como atributos HTML em elementos. Você pode adicionar um valor de string contendo código JavaScript como o valor de um atributo de evento, como onclick, onmouseover, etc.

4

Unidade VII: DOM soulcode.com

Listagem 8: onclick

1 <button onclick="console.log('O botao foi clicado!')">Clique Aqui</button>

Nesse exemplo, quando o botão é clicado, a expressão JavaScript ***console.log(’O botão foi clicado!’)*** é executada.

No entanto, essa abordagem pode dificultar a separação clara entre o có digo HTML e JavaScript, e não é tão flexível ou recomendada para projetos complexos.

**1.4.2 Propriedade de eventos**

Outra forma de definir eventos é usando propriedades de evento diretamente em elementos. As propriedades de evento são associadas diretamente ao elemento e têm como valor uma função de callback.

Listagem 9: Propriedade de eventos

1 const meuElemento = document.getElementById('meu-elemento');

2

3 meuElemento.onclick = function() {

4 console.log('O elemento foi clicado!');

5 };

Nesse exemplo, a propriedade onclick do elemento é definida com uma fun ção de callback anônima. Quando o elemento é clicado, a função é executada. No entanto, essa abordagem substitui qualquer event listener anterior atribuído ao mesmo evento, e pode não ser tão flexível quanto o uso de event listeners.

**1.4.3 EventListener**

Os event listeners são a forma mais comum de definir eventos no JavaScript. Com os event listeners, você pode adicionar uma função de callback para ser executada quando um evento específico ocorre em um elemento alvo.

Listagem 10: EventListener

1 const meuElemento = document.getElementById('meu-elemento');

2

3 function handleClick(e) {

4 console.log('O elemento foi clicado!');

5 console.log(e.target); // referencia do elemento clicado

6 }

7

8 meuElemento.addEventListener('click', handleClick);

9

10 // Mais tarde podemos remover

11 meuElemento.removeEventListener('click', handleClick);

Podemos também definir listeners utilizando arrow functions. 5

Unidade VII: DOM soulcode.com

**1.5 Mais sobre DOM**

DOM é um assunto extenso e sempre há mais para aprender sobre. O ob jetivo deste material é dar direcionamento acerca do assunto. Dessa forma recomendamos a consulta da documentação do MDN Web Docs.

6