# JavaScript: Fundamentos para Desenvolvimento Web Interativo - Turma 2025B

## 4.2 Acessando elementos

Acessar elementos do Document Object Model (DOM) usando JavaScript é uma tarefa comum e fundamental para interagir dinamicamente com uma página da web.O DOM fornece várias formas para que os desenvolvedores encontrem e trabalhem com elementos HTML por meio de suas APIs. Vamos explorar os métodos mais comuns e eficazes para acessar elementos do DOM em JavaScript.

### getElementById

Você pode selecionar um elemento específico usando seu ID usando o método **getElementById**. Como o nome sugere, ele retorna o elemento que possui o atributo ID correspondente ao argumento passado. Este método retorna exatamente um elemento ou null se nenhum elemento com o ID especificado for encontrado. A sintaxe para getElementById é simples, conforme quadro abaixo:

```
const elemento = document.getElementById('idDoElemento');
```

Onde '**idDoElemento**' é uma string que corresponde ao atributo **id** do elemento HTML que você deseja acessar. Se nenhum elemento com o ID especificado for encontrado, **getElementById** retornará **null**.

Suponha que você tenha um elemento HTML como este demonstrado no quadro abaixo:

```
<div id="meuDiv">Algum conteúdo aqui</div>
```

Você pode acessar este elemento no JavaScript da seguinte maneira, conform quadro abaixo:

```
const div = document.getElementById('meuDiv');
console.log(div.textContent); // Saída: Algum conteúdo aqui
```

Para facilitar a explicação, vamos utilizar JavaScript inline no HTML. Isso significa que incluiremos o código JavaScript diretamente dentro de um documento HTML, dentro das tags **<script>**. Veja um exemplo sobre o uso do **getElementById** no quadro abaixo:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Javascript</title>
</head>
<body>
  <div id="meuDiv">Algum conteúdo aqui</div>
  <script>
    const div = document.getElementById('meuDiv');
    console.log(div.textContent); // Saída: Algum conteúdo aqui
  </script>
</body>
</html>
```

Veja o resultado no navegador, conforme figura abaixo:

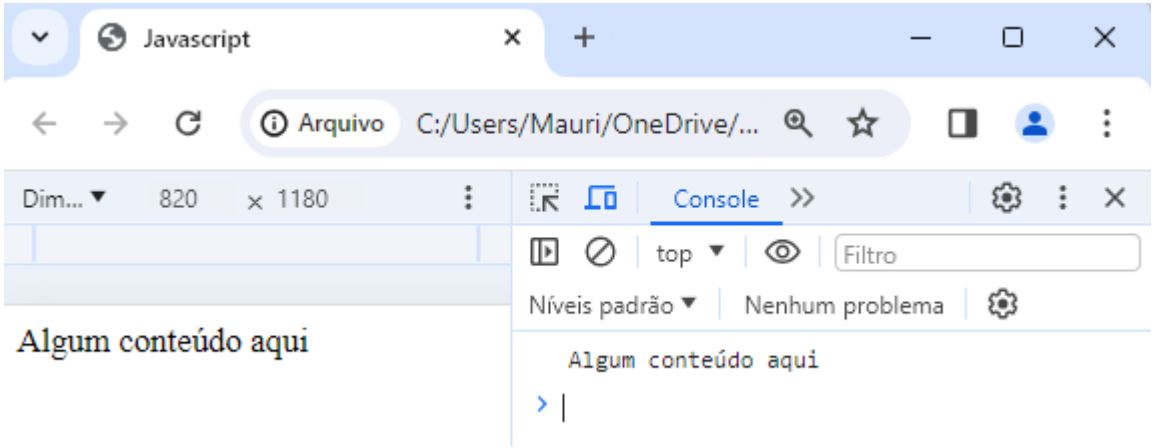

Figura 1 - Exemplo de uso do método getElementById

Descrição da imagem: No navegador, observamos uma página com uma divisão ( ) que tem o id "meuDiv" e contém o texto "Algum conteúdo aqui". Ao carregar a página, o texto "Algum conteúdo aqui" presente na divisão será exibido na aba Console das Ferramentas de Desenvolvedor do navegador. A interface principal da página mostrará apenas a div com o texto, mas o console fornecerá uma visualização adicional do texto contido na divisão.

Uma vez que você tenha acessado um elemento usando **getElementById**, você pode manipulá-lo de várias maneiras, as quais veremos no decorrer deste texto.

## getElementsByClassName

O método **getElementsByClassName** permite selecionar e manipular elementos do DOM que compartilham a mesma classe CSS. Este método retorna uma coleção de todos os elementos que possuem as classes especificadas.

A sintaxe para usar **getElementsByClassName** é mostrada no quadro abaixo:

```
const elementos = document.getElementsByClassName('nome-da-classe');
```

Onde '**nome-da-classe**' é uma string que representa uma ou mais classes para as quais os elementos devem ser selecionados. Classes múltiplas podem ser passadas como uma única string delimitada por espaços.

Para exemplificar o uso deste método, suponha que você tenha vários elementos HTML com a classe **destaque**, conforme quadro abaixo:

```
<div class="destaque">Item 1</div>
<div class="destaque">Item 2</div>
<p class="destaque">Item 3</p>
```

Você pode selecionar todos esses elementos usando **getElementsByClassName**, conforme quadro abaixo:

```
const destaques = document.getElementsByClassName('destaque');
console.log(destaques.length); // Saída: 3
```

**OBS:** A coleção retornada é do tipo **HTMLCollection**, que é semelhante a uma lista de elementos acessíveis por índice. Essa coleção resultante é "ao vivo", o que significa que ela automaticamente se atualiza para refletir as mudanças no DOM. Por exemplo, se elementos que fazem parte da coleção são removidos ou se novos elementos são adicionados ao DOM com a classe especificada, a coleção reflete essas mudanças.

## getElementsByTagName

O método **getElementsByTagName** permite que os desenvolvedores selecionem elementos por suas tags HTML, retornando uma **HTMLCollection** de todos os elementos correspondentes na árvore DOM. Esta coleção é ao vivo, refletindo mudanças no DOM em tempo real.

A sintaxe básica do método **getElementsByTagName** é demonstrada no quadro abaixo:

```
const elementos = document.getElementsByTagName('nomeDaTag');
```

Onde '**nomeDaTag**' é uma string que representa o nome da tag HTML que você deseja selecionar. Por exemplo, para selecionar todos os elementos ** **, você usaria "**div**" como argumento.

Para exemplificar o uso deste método, suponha que você tenha o código HTML conforme quadro abaixo:

```
<div>Primeira div</div>
<p>Parágrafo</p>
<div>Segunda div</div>
```

Para selecionar todos os elementos ** ** e alterar o conteúdo de texto de cada um, você poderia fazer o seguinte:

```
const divs = document.getElementsByTagName('div');
for (let i = 0; i < divs.length; i++) {
  divs[i].textContent = 'Texto atualizado para a div ' + (i + 1);
}
```

Esse código atualizaria o conteúdo de texto de cada ** ** para "**Texto atualizado para a div 1**", "**Texto atualizado para a div 2**", etc.

## querySelector

O método **querySelector** permite que os desenvolvedores selecionem o primeiro elemento que corresponde a um determinado seletor CSS. Lembre-se que esses seletores CSS são usados para direcionar elementos específicos com base em suas características, como tags, classes, IDs, atributos, entre outros.

A sintaxe para querySelector é demonstrada no quadro abaixo:

```
const elemento = document.querySelector(seletorCSS);
```

Onde **seletorCSS** é uma string que representa um ou mais seletores CSS que identificam o elemento desejado. Por exemplo, você pode usar seletores de classe (**.classe**), seletores de ID (**#id**), seletores de atributo (**[atributo="valor"]**), seletores de pseudo-classe (**:hover**, **:nth-child(n)**, etc.), e combinações desses.

Para exemplificar o uso deste método, suponha que você tenha o seguinte HTML, conforme quadro abaixo:

```
<div class="container">
  <ul>
    <li id="primeiro">Primeiro item</li>
    <li>Segundo item</li>
    <li class="destaque">Terceiro item</li>
  </ul>
</div>
```

Para selecionar o primeiro elemento **<li>** dentro do **<ul>** que tem a classe "**destaque**", você pode usar:

```
const itemDestaque = document.querySelector('ul li.destaque');
console.log(itemDestaque.textContent); // Saída: Terceiro item
```

Devido à sua capacidade de usar seletores CSS complexos, **querySelector** proporciona uma grande flexibilidade e precisão na seleção de elementos do DOM.

## querySelectorAll

O método **querySelectorAll** permite os desenvolvedores selecionarem todos os elementos que correspondem a um determinado seletor CSS, retornando uma lista estática de nós, conhecida como **NodeList**. Neste contexto, enquanto o método **querySelector** retorna apenas o primeiro elemento correspondente, **querySelectorAll** retorna todos os elementos que correspondem ao grupo de seletores especificado.

A sintaxe básica de querySelectorAll é demonstrada no quadro a seguir:

```
const elementos = document.querySelectorAll(seletorCSS);
```

Onde **seletorCSS** é uma string que representa um ou mais seletores CSS que podem ser combinados. Esses seletores são usados para encontrar elementos da mesma maneira que você faria em uma folha de estilo CSS.

Para exemplificar o uso deste método, suponha que você tenha o seguinte HTML:

```
<ul class="lista">
  <li class="item">Item 1</li>
  <li class="item active">Item 2</li>
  <li class="item">Item 3</li>
</ul>
```

Para selecionar todos os elementos **<li>** com a classe **item**, você pode fazer o seguinte:

```
const items = document.querySelectorAll('.lista .item');
items.forEach(item => console.log(item.textContent));
```

Veja o resultado, conforme figura abaixo:

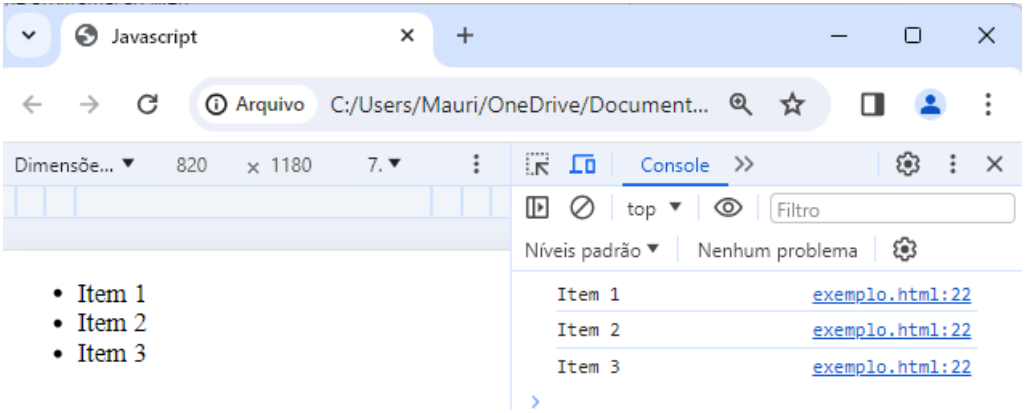


Figura 2: Exemplo de uso do método querySelectorAll

Descrição da imagem: No navegador, observamos uma lista não ordenada (<ul>) com a classe "lista", contendo três itens de lista (<li>). Cada item de lista tem a classe "item", e o segundo item tem uma classe adicional "active". Os itens de lista são "Item 1", "Item 2" e "Item 3". Ao executar o código, o texto de cada item da lista ("Item 1", "Item 2" e "Item 3") será exibido na aba Console das Ferramentas de Desenvolvedor do navegador

**OBS:** Diferente de **getElementsByClassName** e **getElementsByTagName**, que retornam uma **HTMLCollection ao vivo**, a **NodeList** retornada por **querySelectorAll** é estática. Isso significa que as mudanças subsequentes no DOM não afetarão a coleção de elementos retornada.

## children e childNodes

Ao trabalhar com o DOM, você pode precisar acessar os elementos filhos de um elemento pai específico. As propriedades **children** e **childNodes** são duas maneiras pelas quais você pode acessar esses elementos filhos, mas elas têm diferenças importantes em termos do que exatamente elas retornam e como são usadas.

### Propriedade children

A propriedade **children** retorna uma **HTMLCollection** (uma coleção ao vivo) que contém apenas os elementos filhos de um elemento, excluindo todos os nós de texto e comentários. Isso a torna mais útil para muitas tarefas de manipulação do DOM onde apenas elementos são necessários.

Para exemplificar o uso deste método, suponha que você tenha o seguinte HTML:

```
<div id="meuDiv">
    Primeiro texto<p>Primeiro parágrafo</p><!-- Comentário --><p>Segundo parágrafo</p>
</div>
```

Agora, considere o seguinte código JavaScript para acessar **children**:

```
const div = document.getElementById('meuDiv');
console.log(div.children.length); // Isto irá retornar 2
```

Observe que a saída será 2, pois está contando apenas os elementos **<p>**, ignorando o texto e o comentário.

## Propriedade childNodes

A propriedade **childNodes** retorna uma **NodeList** (lista estática) contendo todos os nós filhos de um elemento, incluindo elementos HTML, nós de texto (como espaços em branco, quebras de linha) e comentários. Isso significa que **childNodes** pode incluir mais do que apenas os elementos HTML visíveis que você talvez deseje manipular.

Veja que estas propriedades são usadas para acessar diretamente os filhos de um elemento. A propriedade **children** retorna uma coleção ao vivo de elementos filhos, enquanto **childNodes** retorna todos os nós filhos, incluindo espaços em branco de texto e comentários.

Para exemplificar o uso deste método, suponha que você tenha o seguinte HTML:

```
<div id="meuDiv">
    Primeiro texto<p>Primeiro parágrafo</p><!-- Comentário --><p>Segundo parágrafo</p>
</div>
```

Considere o seguinte código JavaScript para acessar **childNodes**:

```
const div = document.getElementById('meuDiv');
console.log(div.childNodes.length); // Isto pode retornar 5 dependendo do whitespace
```

**OBS:** No HTML, todos os espaços em branco, incluindo espaços, tabulações e quebras de linha, são considerados parte do conteúdo e, portanto, são tratados como nós de texto pelo DOM. Quando você usa **childNodes** para acessar os filhos de um elemento, esses nós de texto de whitespace são incluídos na **NodeList** resultante.

◄ 4.1 Introdução ao Document Object Model (DOM)

Seguir para...

4.3 Modificando elementos ►