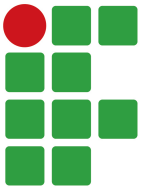

INSTITUTO FEDERAL
Espírito Santo
Campus de Alegre

Apostila Complementar

O DOM (Document Object Model) na Prática

Técnicas Avançadas para o Desenvolvedor Front-End

Desenvolvimento Front-End I

Período: TADS EaD - Semana 9

Autor: Prof. Cleziel Franzoni da Costa

Instituto Federal do Espírito Santo — Campus de Alegre

17 de outubro de 2025

Sumário

1 Introdução

Olá, aluno! Na apostila principal da Semana 9, você foi apresentado ao DOM e aprendeu a realizar as tarefas essenciais: **selecionar elementos** e **alterar seu texto ou estilo**.

Este material complementar aprofunda seu domínio do DOM, apresentando uma **caixa de ferramentas mais completa** para a manipulação de elementos no dia a dia do desenvolvedor. Abordaremos como **selecionar múltiplos elementos**, **alterar atributos**, **criar e remover elementos do zero** e até mesmo **navegar pela "árvore"de nós** com mais eficiência. Dominar estas técnicas é o que diferencia um iniciante de um desenvolvedor front-end proficiente.

Pense neste guia como um mergulho mais fundo no "mapa da página": o objetivo não é apenas ler o mapa, mas **redesenhá-lo dinamicamente** com segurança, performance e clareza.

2 Caixa de Ferramentas de Seleção Avançada

Na apostila principal, focamos em `getElementById` e `querySelector`. Vamos expandir esse conhecimento.

2.1 Selecionando múltiplos elementos com `querySelectorAll()`

Enquanto `querySelector` retorna **apenas o primeiro** elemento que corresponde ao seletor, `querySelectorAll()` é mais poderoso: ele retorna uma **NodeList** (uma coleção de nós) com **todos** os elementos que correspondem.

Isso é indispensável para aplicar a mesma mudança em vários itens, como todos os `<li>` de uma lista ou todos os cards de produto de uma galeria.

2.1.1 Exemplo: Destacar todos os itens de uma lista

```html
<ul>
  <li class="item-lista">Primeiro item</li>
  <li class="item-lista">Segundo item</li>
  <li class="item-lista">Terceiro item</li>
</ul>
```

HTML

```javascript
// 1. Seleciona TODOS os elementos com a classe 'item-lista'
const todosOsItens = document.querySelectorAll('.item-lista');

// 2. A NodeList aceita o método forEach em navegadores modernos,
```

```javascript
// o que nos permite iterar sobre cada elemento encontrado.
todosOsItens.forEach((item) => {
  // 3. Para cada item, aplicamos os estilos.
  item.style.color = '#1a5e3a'; // Verde IFES
  item.style.fontWeight = 'bold';
});
```

JavaScript

! Atenção

A `querySelectorAll()` retorna uma **lista**, não um único elemento. Você não pode aplicar um estilo diretamente (ex: `todosOsItens.style.color`). É **obrigatório** usar um laço de repetição, como o `forEach`, para percorrer a lista e aplicar a alteração a cada item individualmente.

? Dica de Performance e Boas Práticas

Sempre que possível, evite espalhar estilos *inline* pelo seu JavaScript. A melhor prática é ter classes pré-definidas no seu CSS (ex: `.destacado`) e usar o JavaScript apenas para adicionar ou remover essas classes com `item.classList.add('destacado')`. Isso mantém seu CSS no lugar do CSS e seu JS focado no comportamento.

3 Manipulando Atributos e Propriedades

Além do conteúdo textual e do CSS, uma tarefa muito comum é ler e alterar os **atributos** de uma tag HTML, como `src` de uma imagem, `href` de um link ou `disabled` de um botão.

- `elemento.getAttribute('atributo')`: Lê o valor original do atributo, como está no HTML.
- `elemento.setAttribute('atributo', 'valor')`: Cria ou atualiza o valor de um atributo.
- `elemento.removeAttribute('atributo')`: Remove o atributo do elemento.

3.0.1 Exemplo: Trocar uma imagem e desabilitar um botão

```html
<img id="imagem-perfil" src="imagem-antiga.png" alt="Foto de perfil">
<button id="btn-salvar">Salvar Alterações</button>
```

HTML

```javascript
const imagemPerfil = document.getElementById('imagem-perfil');
const botaoSalvar = document.getElementById('btn-salvar');

// 1. Troca o atributo 'src' da imagem
imagemPerfil.setAttribute('src', 'imagem-nova.png');

// 2. Lê o valor do atributo 'alt' e exibe no console
console.log('ALT atual:', imagemPerfil.getAttribute('alt'));

// 3. Desabilita o botão (forma mais comum via propriedade)
// Atributos booleanos como disabled, checked, required
// são mais facilmente controlados por suas propriedades no objeto DOM.
botaoSalvar.disabled = true;

// A forma com setAttribute também funciona:
// botaoSalvar.setAttribute('disabled', 'true');
```

JavaScript

✓ Prática Moderna Recomendada

Para atributos booleanos (`disabled`, `checked`, `required`, etc.), a prática mais comum e limpa é manipular a **propriedade** do objeto DOM diretamente: `elemento.disabled = true;` ou `elemento.checked = false;`.
Outros manipuladores úteis incluem:

- `element.classList`: Para adicionar, remover ou alternar classes CSS.
- `element.dataset`: Para acessar atributos `data-*` (ex: `<div data-id="42">` é acessado via `element.dataset.id`).

4 Criação e Inserção Dinâmica de Elementos

E se o elemento que você quer manipular **ainda não existe** no HTML? O DOM permite criar qualquer elemento do zero, configurá-lo e inseri-lo na página em tempo real.

4.0.1 O Processo de 3 Passos:

1. **Criar:** Use `document.createElement('tag')` para criar o nó em memória.
2. **Configurar:** Defina suas propriedades, como `textContent`, `classList`, ou use `setAttribute`.

3. **Inserir:** Use um método como `append()` em um elemento "pai"já existente para colocar o novo elemento na página.

4.0.2 Exemplo: Adicionar um novo item a uma lista de tarefas

```html
<ul id="lista-tarefas">
  <li>Estudar HTML</li>
  <li>Estudar CSS</li>
</ul>
<button id="btn-adicionar">Adicionar Tarefa</button>
```

HTML

```javascript
const botaoAdicionar = document.getElementById('btn-adicionar');
const listaDeTarefas = document.getElementById('lista-tarefas');

botaoAdicionar.addEventListener('click', () => {
  // 1. CRIAR o elemento <li>
  const novoItem = document.createElement('li');

  // 2. CONFIGURAR o elemento
  novoItem.textContent = 'Estudar JavaScript';
  novoItem.classList.add('item-lista');

  // 3. INSERIR o novo item no final da <ul>
  listaDeTarefas.append(novoItem);
});
```

JavaScript

Ponto Crítico de Segurança

Sempre prefira usar `textContent` ou `innerText` para inserir conteúdo de texto, especialmente se ele vier de um usuário. Evite usar `innerHTML` com dados que você não controla, pois isso abre uma vulnerabilidade de segurança grave conhecida como **Cross-Site Scripting (XSS)**.

Tabela 1: Métodos Modernos de Inserção de Elementos

Método	Descrição
`elementoPai.prepend(novoElemento)`	Insere `novoElemento` como o **primeiro filho**.
`elementoPai.append(novoElemento)`	Insere `novoElemento` como o **último filho**.
`elementoReferencia.before(novoElemento)`	Insere `novoElemento` **antes** do `elementoReferencia`.
`elementoReferencia.after(novoElemento)`	Insere `novoElemento` **depois** do `elementoReferencia`.

5 Navegação e Travessia na Árvore DOM

Às vezes, você não tem um seletor direto para o elemento que deseja, mas conhece sua relação com outro elemento. O DOM permite "caminhar"pela árvore de nós.

- `elemento.parentElement`: Acessa o elemento "pai"direto.
- `elemento.children`: Retorna uma coleção dos elementos "filhos"diretos.
- `elemento.nextElementSibling` / `previousElementSibling`: Acessa o "irmão"seguinte ou anterior.
- `elemento.closest(seletor)`: **Extremamente útil!** Sobe na árvore a partir do `elemento` e retorna o primeiro ancestral que corresponde ao `seletor` CSS fornecido.

5.0.1 Exemplo: Destacar um card ao clicar em um botão interno

```html
<div class="card-produto">
  <h3>Produto A</h3>
  <p>Descrição do produto.</p>
  <button class="btn-comprar">Comprar</button>
</div>
```

HTML

```javascript
// Suponha que temos vários botões com a mesma classe
const botoesComprar = document.querySelectorAll('.btn-comprar');

botoesComprar.forEach(botao => {
  botao.addEventListener('click', function (event) {
    // A partir do botão que foi clicado (event.target)...
    // ...suba na árvore até encontrar o ancestral mais próximo
    // que tenha a classe 'card-produto'.
    const cardPai = event.target.closest('.card-produto');

    // Se encontrou o card, aplique o estilo
    if (cardPai) {
      cardPai.style.backgroundColor = '#e6ffe6';
      cardPai.style.border = '1px solid #28a745';
    }
  });
});
```

JavaScript

✓ Prática Moderna Recomendada

O método `closest()` é muito mais robusto e legível do que fazer múltiplas chamadas a `parentElement` (ex: `this.parentElement.parentElement`). Ele desacopla seu JavaScript de uma estrutura HTML rígida; se você adicionar um `<div>` extra em volta do botão, o código com `closest()` continua funcionando, enquanto o código com `parentElement` quebraria.

6 Conclusão

O DOM é uma API vasta e poderosa. As técnicas abordadas nesta apostila complementar cobrem as operações mais frequentes e essenciais no dia a dia de um desenvolvedor front-end:

- Use `querySelectorAll` para agir em múltiplos elementos de uma vez.
- Use `setAttribute` e propriedades diretas (`.disabled`) para manipular atributos.
- Use o trio **criar** → **configurar** → **inserir** para adicionar conteúdo dinâmico de forma segura.
- Navegue pela árvore com `parentElement`, irmãos e, de preferência, com `closest()` quando precisar encontrar um ancestral específico.

Dominar essas técnicas lhe dará controle total sobre o conteúdo e a estrutura de suas páginas, abrindo caminho para a criação de aplicações web verdadeiramente dinâmicas, acessíveis e seguras.