



Criar e adicionar HTML via JavaScript

Com o Javascript, conseguimos criar elementos e adicioná-los na DOM, tudo isso dinamicamente.

Um meio muito utilizado para criação de elementos é o `createElement()`:

```
const elementoX = document.createElement('nomeDaTag');
```

Você passará o nome da tag desejada para criação entre aspas e dentro do parênteses do método, como a sintaxe acima.

No nosso caso podemos atribuir o elemento criado à uma variável qualquer, para quem sabe utilizarmos em outro momento da nossa aplicação.

Um exemplo é adicionar um texto qualquer no clique de determinado botão:

```
function criacaoDeElementos() {  
  const section = document.querySelector('section');  
  const div = document.createElement('div');  
  const paragrafo = document.createElement('p');  
  let texto = "texto qualquer para exemplo";  
  
  paragrafo.append(texto);  
  div.append(paragrafo);  
  section.append(div);  
}  
  
const btn = document.querySelector('button');  
btn.addEventListener('click', criacaoDeElementos);
```

Resultado:

Clique aqui para gerar um novo elemento

texto qualquer para exemplo

Basicamente criamos uma `div` e um `parágrafo`, após atribuímos um texto qualquer. Também buscamos os elementos na DOM para inserirmos a div criada com o parágrafo dentro de uma `section`. Por fim, adicionamos a função no clique do botão.

Para adicionar filhos em um determinado elemento pai utilizamos o método `appendChild()`:

Sua sintaxe é:

```
var filho = elemento.appendChild(filho);  
  
// filho: elemento a ser adicionado no elemento pai  
// elemento: elemento pai
```

Algumas particularidades:

- Caso este `filho` em questão já estiver dentro de uma `estrutura existente`, ele será retirado de sua posição e movido para a nova posição;
- Um nó `não pode estar em dois locais ao mesmo tempo`, portanto, o processo irá primeiro remover o nó do `pai` em questão para depois adicioná-lo ao `novos` 'pai';

- Caso queira ser feito uma inserção em dois locais precisamos primeiro utilizar `node.cloneNode`, salvando dentro de uma variável e depois adicionar esta variável ao novo 'pai';

Podemos utilizá-lo diretamente no nosso elemento body, por exemplo:

```
var elementoFilho = document.createElement('p');
document.body.appendChild(elementoFilho);
```

Nesse caso não irá renderizar nenhuma informação, pois o elementoFilho não possui nada. Para colocarmos uma frase por exemplo, podemos utilizar outro método chamado `innerText()` diretamente no elementoFilho:

```
elementoFilho.innerText = 'Texto qualquer.'
```

Texto qualquer.

Para inserirmos algum filho em dois lugares diferentes usamos o método `cloneNode()`:

Sua sintaxe é:

```
var duplicado = node.cloneNode();
```

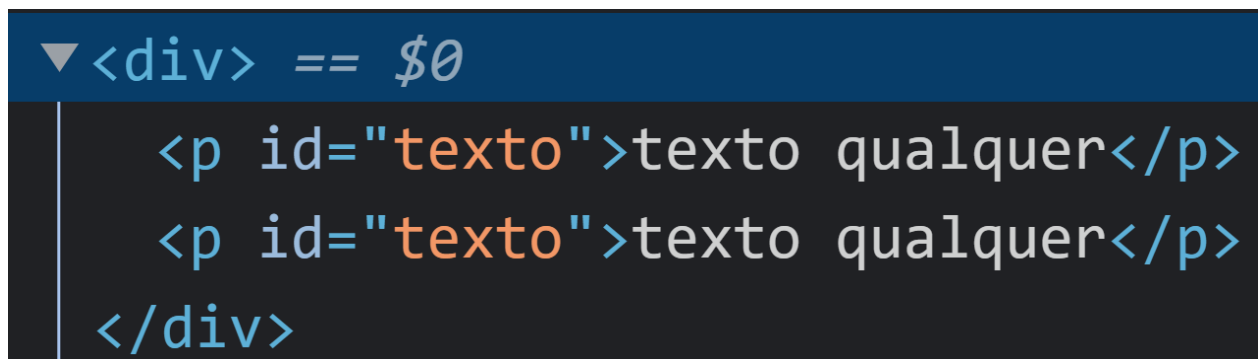
Criamos uma variável e então duplicamos o nó em questão, por exemplo:

```
<div>
  <p id="texto">texto qualquer</p>
</div>
```

```
let div = document.querySelector('div');
let p = document.getElementById("texto");
let clone = p.cloneNode(true);

div.append(clone);
```

Caso você verifique o html diretamente no navegador, perceberá que esse método duplica o elemento por inteiro, ou seja, além de duplicar seu valor ele duplicará seus atributos, como o id:



```
▼ <div> == $0
  <p id="texto">texto qualquer</p>
  <p id="texto">texto qualquer</p>
</div>
```

Para removermos um filho de determinado nó pai utilizamos o método `removeChild()`:

Sua sintaxe:

```
var elementoRemovido = elementoPai.removeChild(filho);
elementoPai.removeChild(filho);
```

Esse método é responsável também por guardar o valor "removido". No exemplo acima podemos acessar esse valor através da variável 'elementoRemovido'.

Vamos imaginar que precisamos remover algum elemento do nosso HTML para realizarmos atualizações de layout dinamicamente por exemplo, ficará parecido com:

```
let divPai = document.getElementById('pai');
let elementoFilho = document.querySelector('#filho');

let elementoRemovido = divPai.removeChild(elementoFilho);
```

Nesse caso apenas buscamos os elementos pai e filho na DOM e removemos normalmente.

Porém, temos outras possibilidades de remoção, como:

- Removendo o filho sem precisar de um pai:

```
let filho = document.getElementById('filho');
if(filho.parentNode){
  filho.parentNode.removeChild(filho);
}
```

Passo 1: Primeiramente buscamos o elemento filho na DOM;

Passo 2: Verificamos se o mesmo possui um pai com a sintaxe `elemento.parentNode` ;

Passo 3: Caso possua, utilizamos a mesma sintaxe para remover o elemento filho.

O `parentNode` é o nó parente do nó referenciado. Ele nos retorna um parente do elemento, que nesse caso é o pai.

- Removendo todos os filhos de um pai:

```
var elementoPai = document.getElementById('pai');
while(elementoPai.firstChild){
  elementoPai.removeChild(elementoPai.firstChild);
}
```

Passo 1: Buscamos o elemento pai na DOM;

Passo 2: Usamos o laço while para executar o passo 3 caso a condição (existe um elemento filho) seja verdadeira;

Passo 3: Removemos então o primeiro elemento filho retornado.

Basicamente o laço While executa um bloco específico enquanto a condição de teste for verdadeira. Essa condição é avaliada verdadeira ou falsa antes de ser executada.

Agora imagine que um certo cliente pediu para você transferir determinadas informações de seu site no clique de um botão. Algo simples e fácil, como mudar a posição de dois textos por exemplo. Você poderá realizar essa tarefa utilizando os métodos estudados até então.

Posição 1:

Texto 1

Posição 2:

Texto 2

Mover textos

Resolução:

```
<!-- HTML: -->

<main>
  <div id="pai">
    <h1 id="pos1">Posição 1:</h1>
    <p id="filho1">Texto 1</p>
    <h1 id="pos2">Posição 2:</h1>
    <p id="filho2">Texto 2</p>
  </div>
  <button>Mover textos</button>
</main>
```

```
#filho1 {
  color: red;
}

#filho2 {
  color: green;
}
```

```
// Javascript:

var posicao1 = document.getElementById('pos1');
var posicao2 = document.getElementById('pos2');

var texto1 = document.getElementById('filho1');
var texto2 = document.getElementById('filho2');

var divPai = document.querySelector('div');

function mudarPosicao() {
  let remocao1 = divPai.removeChild(texto1);
  let remocao2 = divPai.removeChild(texto2);

  divPai.appendChild(posicao1);
  divPai.appendChild(remocao2);
  divPai.appendChild(posicao2);
  divPai.appendChild(remocao1);
}

const btn = document.querySelector('button');
btn.addEventListener('click', mudarPosicao);
```

- 1: Buscamos todos os elementos da DOM que utilizaremos;
 - 2: Criamos uma função chamada mudarPosicao, responsável por realizar todas as nossas mudanças;
 - 3: Começamos pela remoção dos textos de seu pai (div);
 - 4: Depois "reconstruimos" os títulos em questão, seguido dos textos invertidos de posição;
 - 5: Com a função criada, devemos atribuí-la a algum evento, então a chamamos no clique do botão;
- Por fim, fica parecido com:

Posição 1:

Texto 2

Posição 2:

Texto 1

Mover textos

Bibliografias:

- <https://www.w3.org/TR/DOM-Level-3-Core/core.html#ID-1734834066>;
- <https://developer.mozilla.org/pt-BR/docs/Web/API/Node/firstChild>;
- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/while>;
- <https://developer.mozilla.org/pt-BR/docs/Web/API/Node/parentNode>;
- <https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement>;