MATERIAL EXTRA

Capturando elementos do DOM

growdev

Document

O item `document` permite um acesso fácil e rápido a todos os **elementos de uma página HTML** através da interface DOM. Apesar de ser filho de window, o objeto `document` também possui seu atalho. Por isso, é possível acessá-lo a partir do comando `window.document` ou apenas `document`.

Os elementos do HTML são mapeados dentro de `document` como nós de uma **árvore**, da mesma forma em que são estruturados no documento. Cada nó é chamado de `element` dentro da estrutura do DOM. Logo, o element raiz do documento é a tag `<html>`. As tags `<body>` e `<head>` são os filhos subsequentes. Logo após, o restante do HTML é mapeado.

Com a interface do DOM e os recursos presentes no objeto `document`, é possível selecionar elementos e armazená-los em variáveis do Javascript, bem como alterar a árvore com a criação de elementos ou alteração dos já existentes. Inclusive, é possível **trocar completamente** o conteúdo de uma página a partir do código Javascript.

Por isso, a quantidade de possibilidades que surgem pela combinação DOM + JS é incrivelmente grande e serviu como base para a criação de inúmeros frameworks e bibliotecas front-end, como é o caso do React.js, o Angular e o Vue – os mais populares atualmente.

Capturando Elementos HTML

getElementById

A forma mais básica de recuperar um elemento dentro do DOM é usando o comando `getElementById` de `document`. A função recebe como parâmetro um ID que deve corresponder ao ID do elemento HTML alvo.

Considere o HTML abaixo onde temos a definição de uma dentro de `body` que contém duas tags: <h1> e <p>. Enquanto <h1> recebe o ID "title", <p> possui "content" como identificador único. O script JS é carregado após a definição dos elementos.

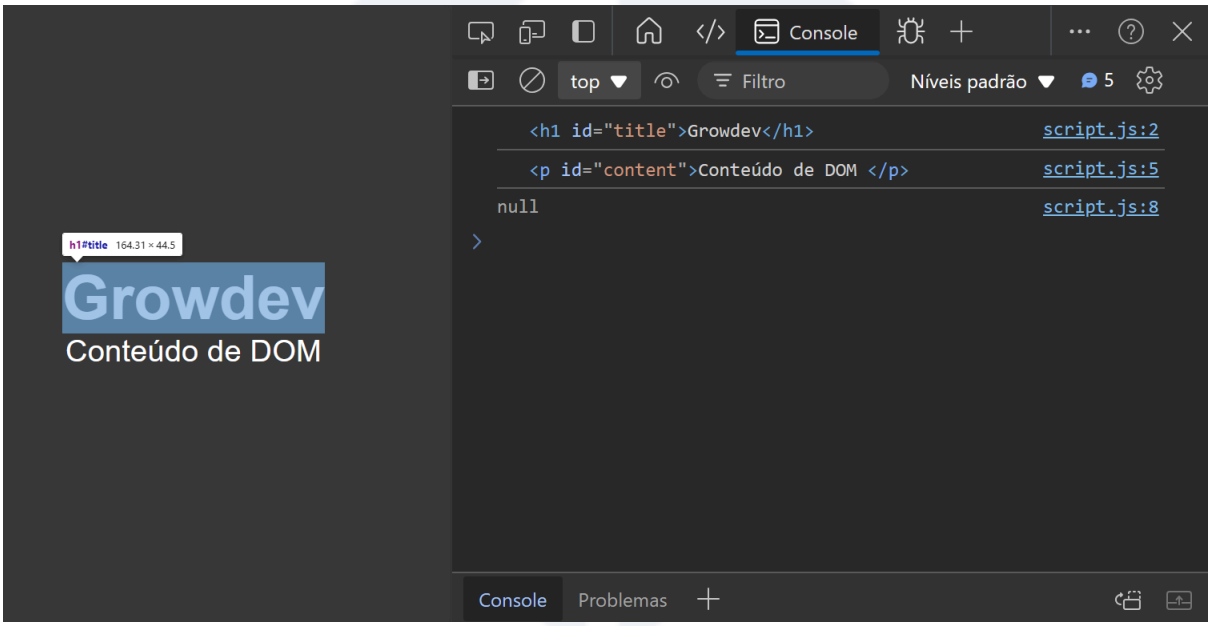
```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - DOM</title>
  </head>
  <body>
      <div>
          <h1 id="title">Growdev</h4>
          <p id="content">Conteúdo de DOM</h4>
      </div>
    <script src="./script.js"></script>
  </body>
</html>
```

Usando a função do DOM `getElementById`, é possível selecionar e armazenar em variáveis cada um dos elementos com determinado **ID**. No exemplo abaixo, ambas as tags <h1> e <p> são selecionadas a partir de seu ID e são mostradas no console.

```
const h1Element = document.getElementById("title")
console.log(h1Element)

const conteudo = document.getElementById("content")
console.log(conteudo)

const elementInexistente = document.getElementById("id-inexistente")
console.log(elementInexistente)
```



No exemplo acima, há uma consulta a um ID inexistente no documento. O retorno de um `getElementById` quando o elemento não existe é `null`. Por outro lado, os elementos selecionados e encontrados são exibidos no console como objetos, descritos pelo nome da tag seguido do identificador único. Os objetos possuem as características do elemento e são nomeados conforme o tipo do elemento.

getElementsByClassName

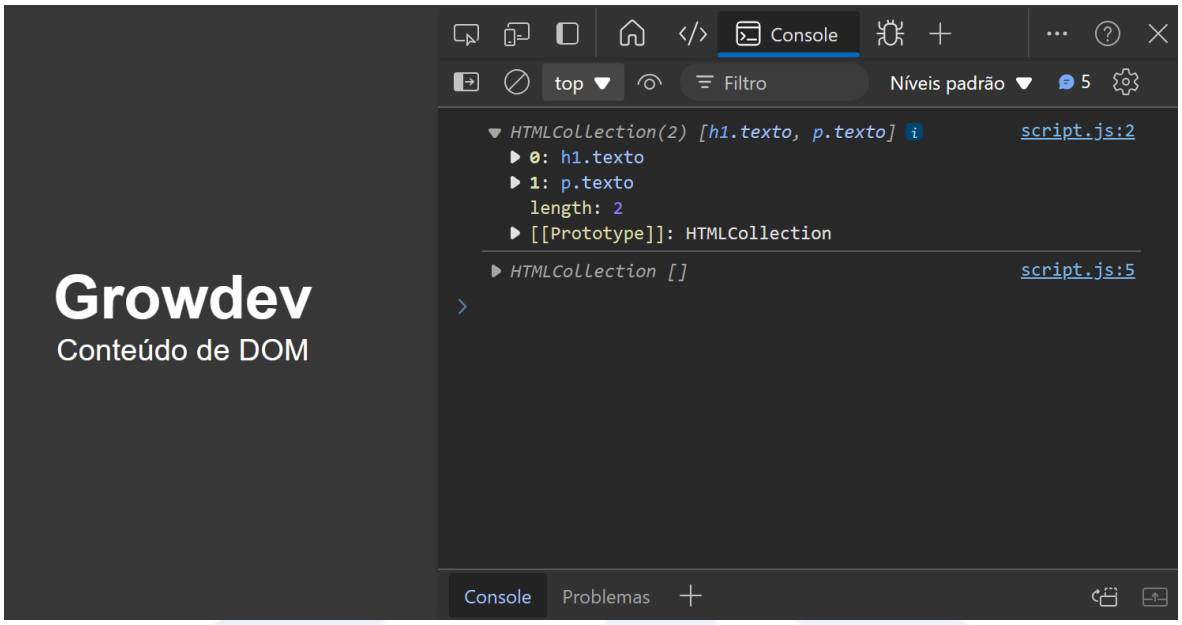
Outra forma de capturar elementos HTML é através das **classes**. A interface DOM define a função `getElementsByClassName` com este objetivo, sendo necessário passar o nome da classe como parâmetro. Neste caso, o valor retornado é uma lista de elementos – já que o HTML permite mais de um elemento com a mesma classe.

Modificando o exemplo de HTML anterior, definimos que tanto `<h1>` quanto `<p>` possuem a classe "texto". Assim, basta selecionarmos ambos em uma mesma lista através do comando `getElementsByClassName`.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - DOM</title>
  </head>
  <body>
      <div >
          <h1 class="texto">Growdev</h4>
          <p class="texto">Conteúdo de DOM</h4>
      </div>
    <script src="./script.js"></script>
  </body>
</html>
```

```
const textList = document.getElementsByClassName("texto")
console.log(textList)

const inexistente = document.getElementsByClassName("class-inexistente")
console.log(inexistente)
```



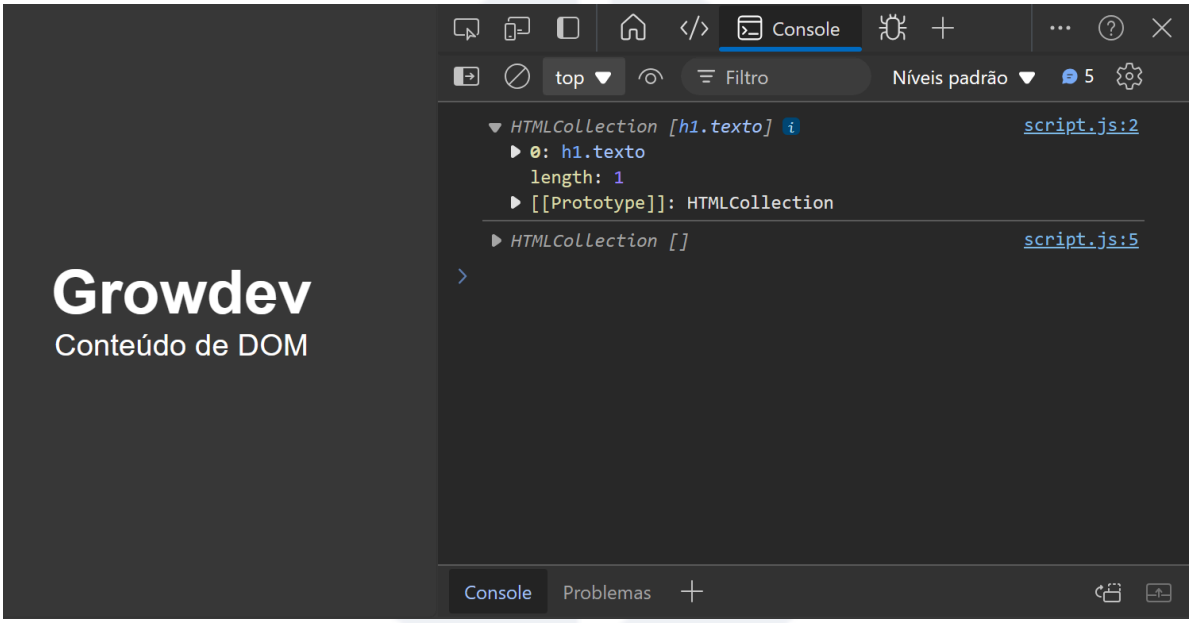
O retorno do primeiro comando no código seleciona os elementos do documento que possuem a `class` "texto". O retorno é uma **lista** com tipo `HTMLCollection`, sendo que cada item é um `element`. Já o segundo comando seleciona uma `class` inexistente e retorna um array vazio. É importante notar que o retorno não é `null` como no caso de consulta por ID.

getElementsByTagName

A seleção de elementos através do **nome da tag** é feita pela função `getElementsByTagName`. Seu funcionamento é similar ao da consulta por `class`, sendo que o parâmetro de entrada é uma string com o nome da tag alvo. O retorno é também uma **lista** dos elementos que foram encontrados.

```
const h1Elements = document.getElementsByTagName("h1")
console.log(h1Elements)

const elementInexistente = document.getElementsByTagName("div")
console.log(elementInexistente)
```



Usando o mesmo HTML usado no exemplo anterior, o script acima seleciona todas as tags <h1> do código. Como temos apenas um <h1>, o retorno é uma lista contendo apenas um `element`. Já o retorno de tags inexistentes também é uma lista vazia.

querySeletor

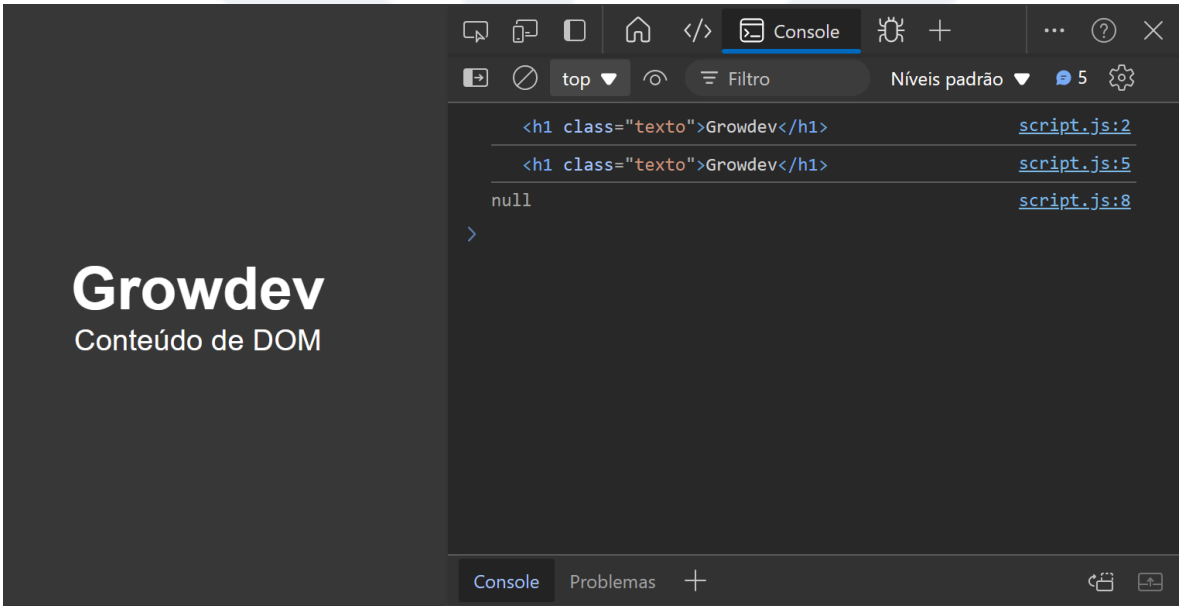
A função `querySelector` permite a **consulta de elementos** usando a sintaxe de **seletores do CSS**. Assim é possível consultar IDs, classes e tags usando o mesmo comando. O comando também permite o uso de todas as combinações de seletores, tornando-se um recurso extremamente útil.

Porém, o `querySelector` possui uma característica importante: ainda que a consulta feita resulte em mais de um elemento, **apenas o primeiro encontrado** no documento é retornado. Por isso seu retorno não é uma lista, mas sim um elemento. Considere ainda o HTML do exemplo anterior para o script abaixo.

```
const h1QuerySelector = document.querySelector("h1")
console.log(h1QuerySelector)

const textoQuerySelector = document.querySelector(".texto")
console.log(textoQuerySelector)

const elementInexistente = document.querySelector("#id-inexistente")
console.log(elementInexistente)
```



O primeiro comando selecionou o único `<h1>` presente no documento e o retornou. Já o segundo comando faz a busca pela classe “texto”. Ainda que existam dois elementos no HTML usado como exemplo, apenas o

primeiro `element` com esta classe foi retornado: o mesmo <h1>. Já quando a busca não encontra qualquer elemento, o retorno é `null`.

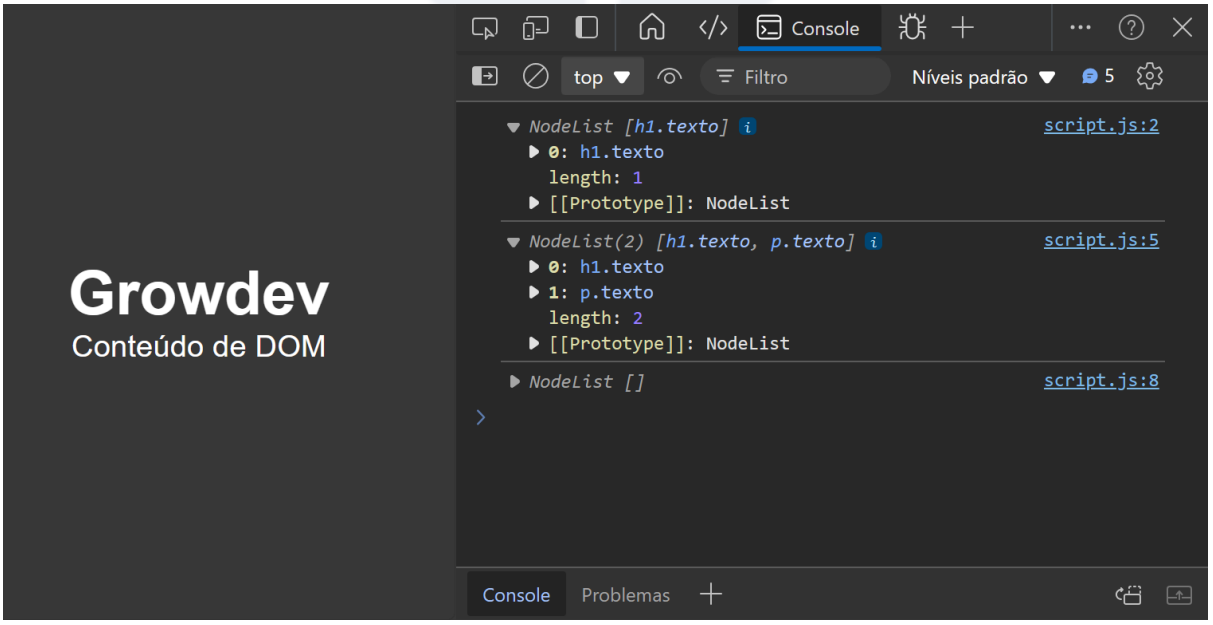
querySeletorAll

O `querySelectorAll` possui o mesmo funcionamento de `querySelector` quanto aos parâmetros de entrada, mas retorna uma **lista** com **todos os elementos encontrados** na busca. Assim é possível usar os seletores CSS para listar vários elementos.

```
const h1QuerySelector = document.querySelectorAll("h1")
console.log(h1QuerySelector)

const textoQuerySelector = document.querySelectorAll(".texto")
console.log(textoQuerySelector)

const elementInexistente = document.querySelectorAll("#id-inexistente")
console.log(elementInexistente)
```



O código acima usa o `querySelectorAll` com os mesmos parâmetros usados no exemplo anterior de `querySelector`. Perceba que o retorno é sempre uma lista. No primeiro comando, a lista possui apenas a tag <h1>. Já na busca pela classe "texto", os dois elementos com esta classe foram retornados. Quando a busca não encontra elementos, uma lista vazia é retornada.

NodeList vs HTMLCollection

NodeList e HTMLCollection são dois tipos de coleções utilizadas para manipular elementos no DOM (Document Object Model), mas possuem diferenças importantes que afetam como e quando devem ser usados.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - DOM</title>
  </head>
  <body>
      <div >
          <h1 class="texto">Growdev</h4>
          <p class="texto">Conteúdo de DOM</h4>
      </div>
    <script src="./script.js"></script>
  </body>
</html>
```

O **NodeList** é uma coleção que pode conter diferentes tipos de nós do DOM, como elementos HTML, nós de texto ou até comentários. Ele é geralmente retornado por métodos como **`querySelectorAll()`**, que permite selecionar múltiplos elementos com base em um seletor CSS.

Uma característica marcante do NodeList é que, na maioria dos casos, ele é **estático**, ou seja, **ele não reflete mudanças feitas posteriormente no DOM**. Se novos elementos forem adicionados ou removidos após a captura do NodeList, essa coleção não será atualizada automaticamente. Além disso, o NodeList é iterável, o que significa que você pode percorrê-lo diretamente utilizando métodos como `forEach`, facilitando bastante seu uso para aplicar operações em múltiplos elementos de forma prática.

```
const items = document.querySelectorAll(".texto") // Retorna um NodeList

items.forEach((item) => {
  console.log(item.innerText)
})
```

Já o **HTMLCollection** é uma coleção que contém exclusivamente elementos HTML e é retornado por métodos como **`getElementsByClassName()`** ou **`getElementsByTagName()`**. Diferentemente do NodeList, o HTMLCollection é **dinâmico**, o que significa que **ele é atualizado automaticamente para refletir qualquer mudança no DOM**. Por exemplo, se um elemento correspondente ao critério de seleção for adicionado ou removido após a captura do HTMLCollection, essa mudança será imediatamente refletida na coleção. No entanto, o HTMLCollection não é iterável de maneira direta; para percorrê-lo, é necessário convertê-lo em um array ou usar estruturas de laço mais tradicionais, como **`for`**.

```
const items = document.getElementsByClassName("texto") // Retorna um HTMLCollection

Array.from(items).forEach((item) => {
  console.log(item.innerText)
})
```

A escolha entre NodeList e HTMLCollection depende do contexto. Se a prioridade for capturar elementos de forma estática para processá-los independentemente de alterações subsequentes no DOM, o NodeList é a melhor opção, especialmente devido à sua facilidade de iteração. Por outro lado, se a necessidade for lidar com uma coleção que reflita dinamicamente as alterações no DOM, o HTMLCollection será mais adequado.

Portanto, entender a diferença entre essas coleções é essencial para manipular o DOM de maneira eficiente e escolher a abordagem mais apropriada para cada situação.

Resumo das Diferenças

Característica	NodeList	HTMLCollection
Conteúdo	Nós do DOM (elementos, textos, etc.)	Apenas elementos HTML
Atualização automática	Geralmente estático	Dinâmico (atualiza automaticamente)
Iterabilidade	Diretamente iterável (`forEach`)	Não iterável diretamente

Qual usar?

- Use **NodeList** quando quiser capturar elementos e iterar sobre eles facilmente.
- Use **HTMLCollection** quando precisar de uma coleção que se atualize automaticamente com mudanças no DOM.

REFERÊNCIAS

1. Introdução ao DOM - APIs da Web | MDN
2. JavaScript DOM Tutorial
3. Document - Web APIs | MDN

BONS ESTUDOS

growdev