MATERIAL EXTRA

Manipulando Atributos

growdev

Manipulando Atributos de Elementos

Após a seleção dos elementos da DOM, é possível **manipular** cada um deles usando métodos específicos. Cada `element` representa um **componente HTML** no documento, como uma tag `<div>`, uma `<body>` ou um `<p>`. Com a interface de `element` do DOM, é possível manipular os elementos e obter informações sobre eles.

Para os exemplos abaixo, considere o seguinte HTML:

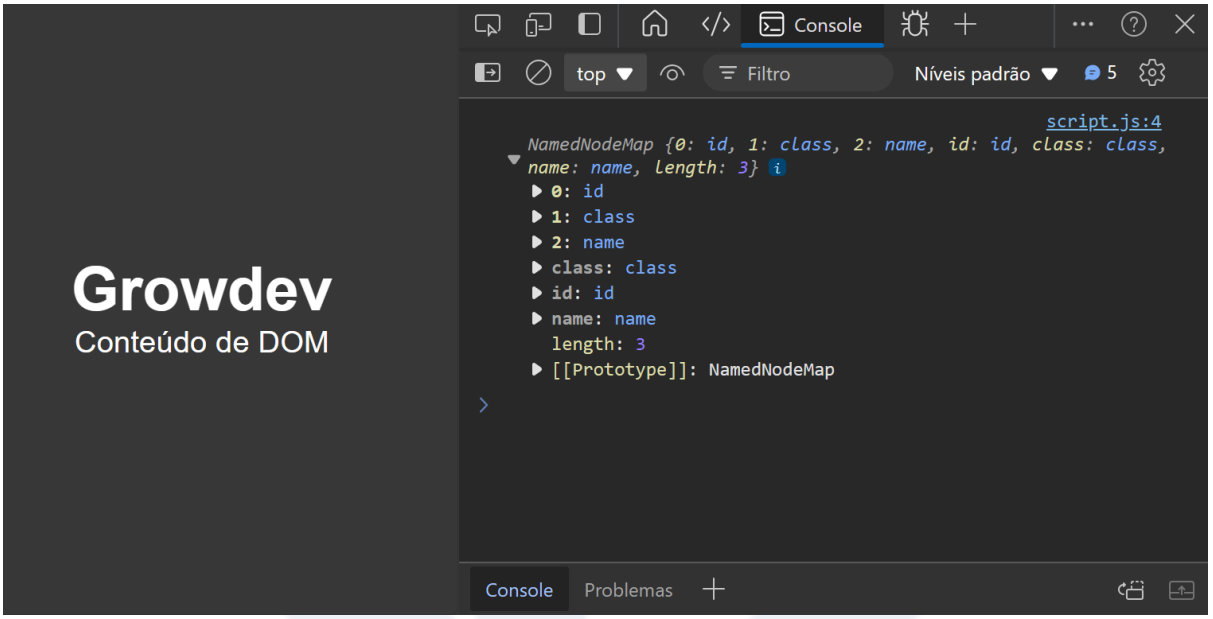
```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - DOM</title>
  </head>
  <body>
      <div>
          <h1 id="title" class="texto">Growdev</h4>
          <p class="texto">Conteúdo de DOM</h4>
      </div>
    <script src="./script.js"></script>
  </body>
</html>
```

setAttribute

O comando `setAttribute` permite adicionar um **novo atributo** a um elemento selecionado. Seus parâmetros de entrada são, em ordem, o nome do atributo e o valor. Caso o atributo já exista, seu conteúdo será alterado.

```
const h1Element = document.getElementById("title");
h1Element.setAttribute("name", "abc");

console.log(h1Element.attributes);
```



O código acima adicionou o `attribute` "name" a tag com ID "title", que é um `<h1>`. O elemento alvo já continha dois atributos no documento carregado: `ID` e `class`. Com a nova adição, o elemento passou a ter três atributos.

A propriedade que armazena os atributos de um elemento é a `attributes` dentro do próprio `element`. Para listar no console, basta fazer o log desta propriedade. É preciso cuidar a estrutura da propriedade `attributes` que **NÃO é uma lista**, mas sim um **objeto** contendo cada um dos atributos.

Adicionalmente à adição, existem métodos para a **remoção** de um atributo já presente no elemento (`removeAttribute`) e outro para forçar

um **toggle** (`toggleAttribute`) que, por sua vez, remove o atributo caso ele exista ou adiciona caso não exista.

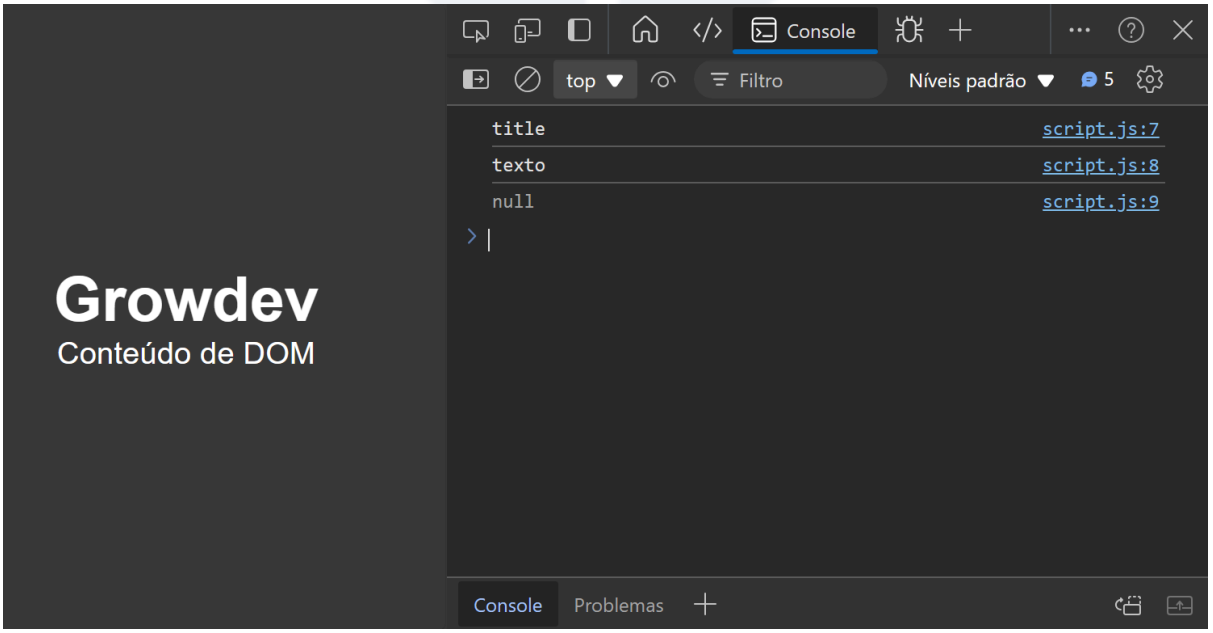
getAttribute

Existem duas formas de **obter** o valor de um atributo via interface `element`. O comando `getAttribute` é o caminho mais simples, sendo uma função que recebe o nome do atributo como parâmetro e retorna o seu valor.

```
const h1Element = document.getElementById("title");

const h1Id = h1Element.getAttribute("id");
const h1Class = h1Element.getAttribute("class");
const h1InexAttr = h1Element.getAttribute("inexistent-attr");

console.log(h1Id);
console.log(h1Class);
console.log(h1InexAttr);
```

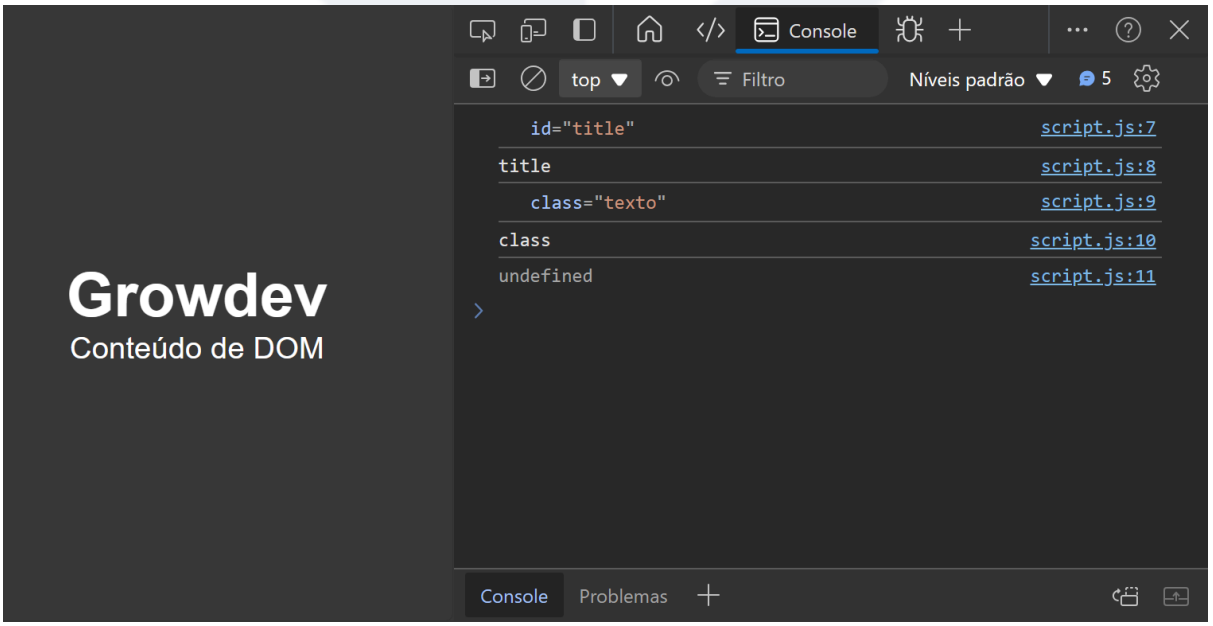


O retorno da função `getAttribute` é sempre o valor do atributo como string. Caso o atributo não exista no elemento, o retorno é `null`. Opcionalmente, pode-se obter um atributo usando o objeto `attributes` presente nos elementos.

```
const h1Element = document.getElementById("title");

const h1Id = h1Element.attributes[0];
const h1Class = h1Element.attributes["class"];
const h1InexAttr = h1Element.attributes["inexistent-attr"];

console.log(h1Id);
console.log(h1Id.value);
console.log(h1Class);
console.log(h1Class.name);
console.log(h1InexAttr);
```



Neste caso, note que o retorno passa a ser um objeto chave-valor e deixa de ser apenas uma string com o valor do atributo. Para acessarmos o valor do atributo, devemos usar a propriedade `value` do atributo. Já para acessarmos o nome, a propriedade é `name`.

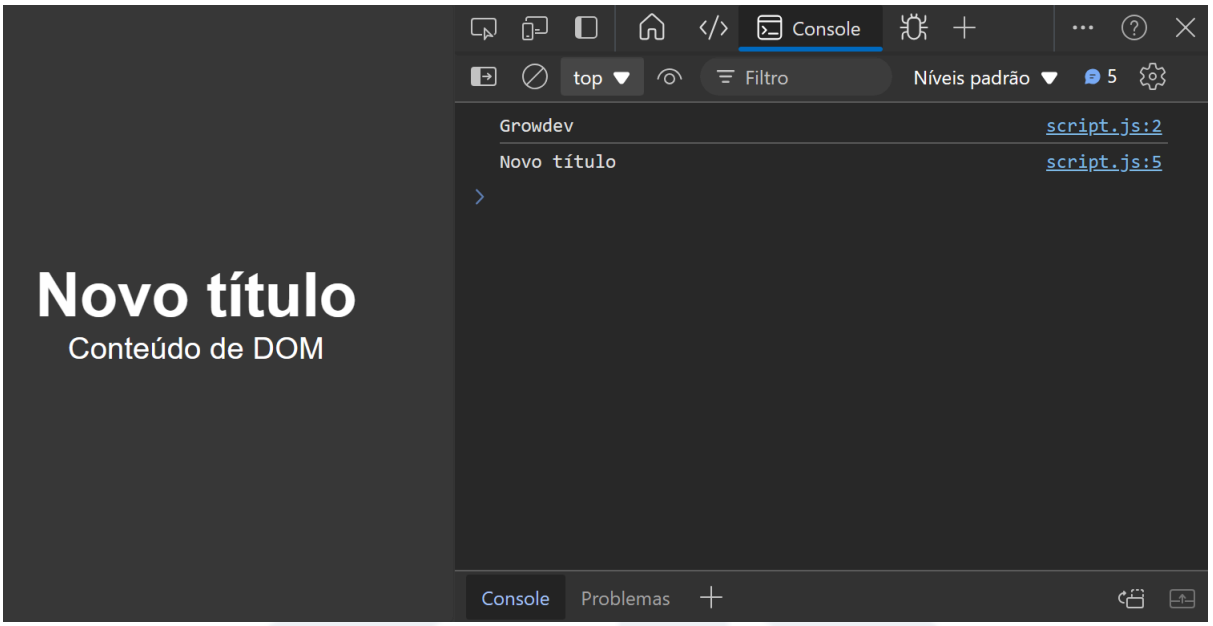
A diferença para `getAttribute` também pode ser percebida no retorno de uma atributo não existente. Enquanto `getAttribute` retorna `null` neste caso, o acesso via `attributes` retorna `undefined`. Além disso, a busca usando o objeto `attributes` também permite o uso de índices numéricos, como se fosse uma lista.

innerHTML

A propriedade `innerHTML` é usada para **obter o conteúdo HTML** de um elemento. Ou seja, tudo que está dentro da tag selecionada pode ser obtido e recuperado para o Javascript. Além disso, é possível **atribuir** um novo conteúdo ao HTML do elemento usando esta mesma propriedade, a partir da atribuição simples em `innerHTML`.

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

h1Element.innerHTML = "Novo título";
console.log(h1Element.innerHTML)
```



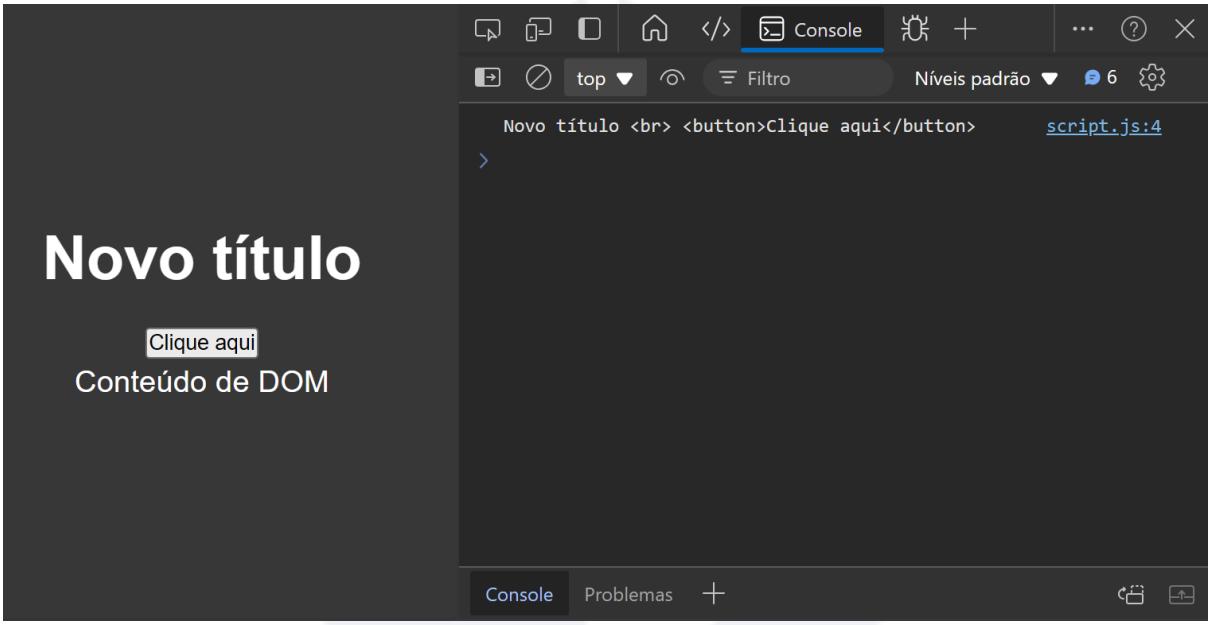
No exemplo acima, o conteúdo inicial da tag <h1> era o texto “Growdev” e por isso a primeira linha do console mostra este conteúdo. Porém um novo valor foi atribuído ao `innerHTML` do elemento logo depois, de modo que a segunda linha do console já mostra o valor atualizado.

Perceba que a mudança feita no `innerHTML` do script foi refletida na própria página, que alterou o título para “Novo título” imediatamente. Isto acontece pois a página é atualizada conforme o DOM é alterado em **tempo real**.

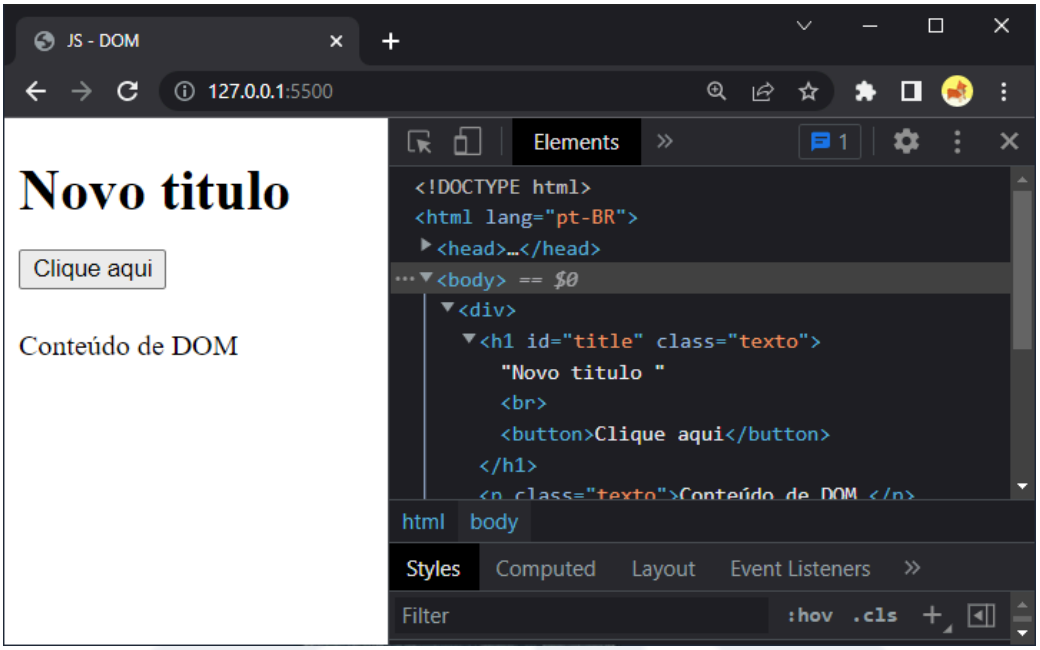
O `innerHTML` não aceita apenas textos na sua atribuição de valores. Com esta interface, podemos inclusive adicionar novas tags HTML dentro do elemento. Ainda assim, o código precisa receber uma string, mesmo que seu conteúdo seja repleto de tags HTML. Observe o exemplo abaixo.

```
const h1Element = document.getElementById("title");

h1Element.innerHTML = “Novo título <br> <button>Clique aqui</button>”;
console.log(h1Element.innerHTML);
```



Neste exemplo, alteramos o `innerHTML` do elemento `<h1>` para que recebesse uma string contendo tags HTML. Adicionamos uma tag `<br>` para gerar uma quebra de linha e uma tag `<button>` para que um botão fosse criado. O resultado na página reflete exatamente os elementos HTML que adicionamos. O retorno de `innerHTML`, porém, se mantém como string.



Note como a estrutura HTML foi alterada para representar o novo `innerHTML` da tag `<h1>` escolhida. O conteúdo de fato fica dentro da tag, conforme o nome da propriedade bem sugere. Além disso, é possível ir alterando a propriedade aos poucos usando **concatenação de strings**, conforme exemplo abaixo. Sendo assim, esta é uma forma prática para criarmos novos elementos no documento.

```
const h1Element = document.getElementById("title");

h1Element.innerHTML = "Novo título <br> <button>Clique aqui</button>";
h1Element.innerHTML += "<br><span>Exemplo de legenda</span>";
```

value

Alguns elementos no HTML possuem a definição do atributo `value`, como é o caso de tags `<input>`. Usando a interface de `element` do DOM, é possível obter os **valores** destes elementos, bem como alterá-los em tempo real.

Para isso, usamos a propriedade `value`. Para este exemplo, considere o HTML abaixo.

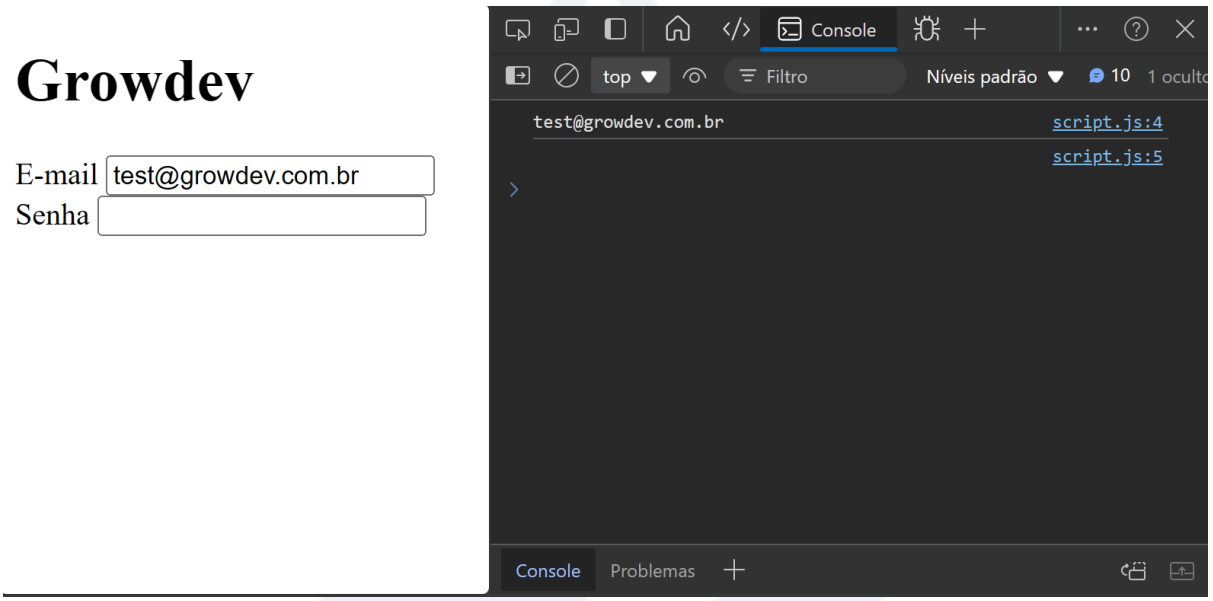
```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - DOM</title>
  </head>
  <body>
      <div>
          <h1 id="title" class="texto">Growdev</h4>
      </div>

      <div>
          <label for="email">E-mail</label>
          <input id="email" type="text" value="test@growdev.com.br">
          <br>
          <label for="senha">Senha</label>
          <input id="senha" type="password">
      </div>
    <script src="./script.js"></script>
  </body>
</html>
```

Note que o exemplo de HTML usa a propriedade `value` para definir um valor inicial ao e-mail, porém a senha vem vazia como default. Usando a propriedade value, podemos então buscar os valores dos inputs do documento e mostrá-los no console.

```
const emailInput = document.getElementById("email");
const senhaInput = document.getElementById("senha");

console.log(emailInput.value);
console.log(senhaInput.value);
```



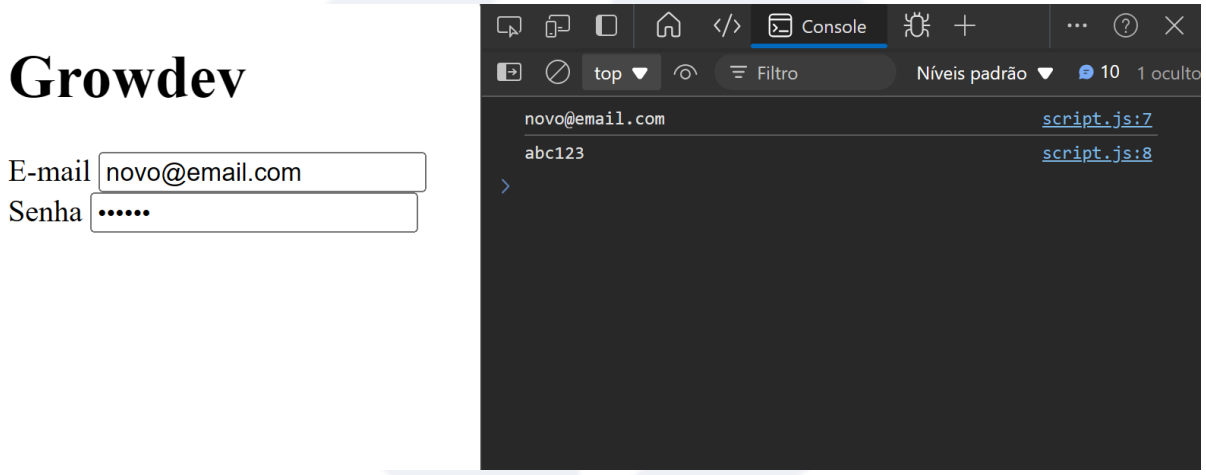
O resultado do script no console apresenta ambos os valores dos `inputs` da tela. Como o e-mail possui valor preenchido, o console o apresenta. Já a senha não possui valor. Neste caso, é importante notar que o valor é uma string vazia – e não `null` ou `undefined` – pois todos os elementos `<input>` possuem `value`. Apenas quando um elemento de fato não possui `value` (p.ex. uma tag `<h1>`) que o retorno é `undefined`.

Além de consultar o valor do atributo `value`, é possível **setar novos valores** através do código. Ou seja, podemos alterar o `<input>` em tempo real. Este recurso é comumente usado para **ajustes** nos valores dos `inputs`, como remoção de caracteres especiais ou controle de tamanho máximo do valor informado.

```
const emailInput = document.getElementById("email");
const senhaInput = document.getElementById("senha");

emailInput.value = "novo@email.com";
senhaInput.value = "abc123";

console.log(emailInput.value);
console.log(senhaInput.value);
```



style

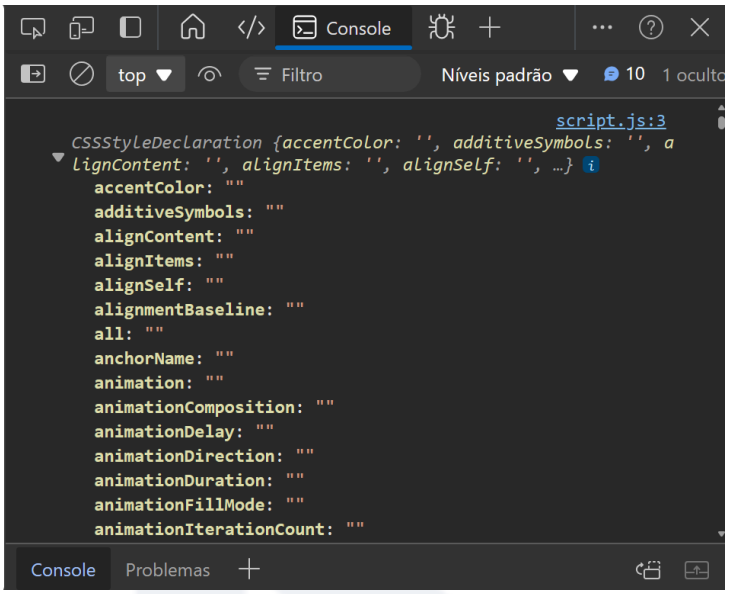
A propriedade `style` é um objeto que contém toda a **estilização CSS** feita para determinado elemento. Com ela, podemos consultar as propriedades e valores atribuídos ao elemento, bem como manipulá-los em tempo real.

```
const emailInput = document.getElementById("email");

console.log(emailInput.style);
```

Growdev

E-mail test@growdev.com.br
Senha

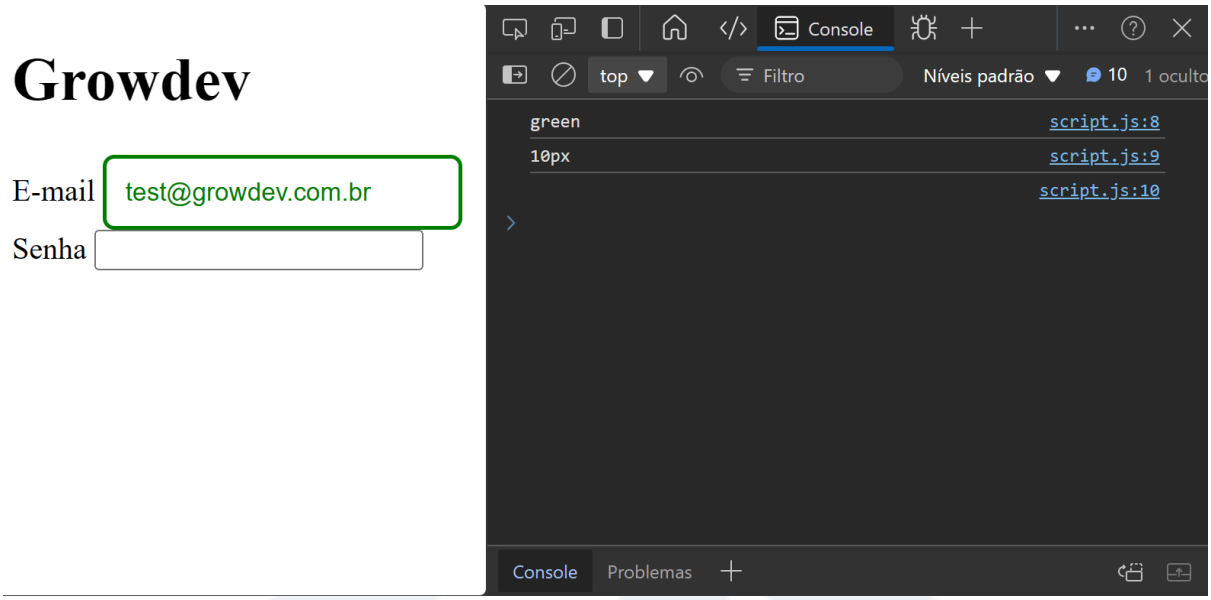


O retorno de `style` é um objeto que contém todas as propriedades CSS disponíveis na interface do DOM, mesmo que o elemento não possua valor atribuído. Portanto, é possível também acessar propriedades específicas do `style` a fim de consultas ou modificações.

```
const emailInput = document.getElementById("email");

emailInput.style.color = "green";
emailInput.style.border = "2px solid green";
emailInput.style.borderRadius = "6px";
emailInput.style.padding = "10px";

console.log(emailInput.style.color);
console.log(emailInput.style.padding);
console.log(emailInput.style.margin);
```



No exemplo acima, o `<input>` de e-mail teve sua estilização alterada através do DOM. A cor do texto, a borda e um `padding` foram setados e a página foi alterada assim que os comandos foram executados no Javascript.

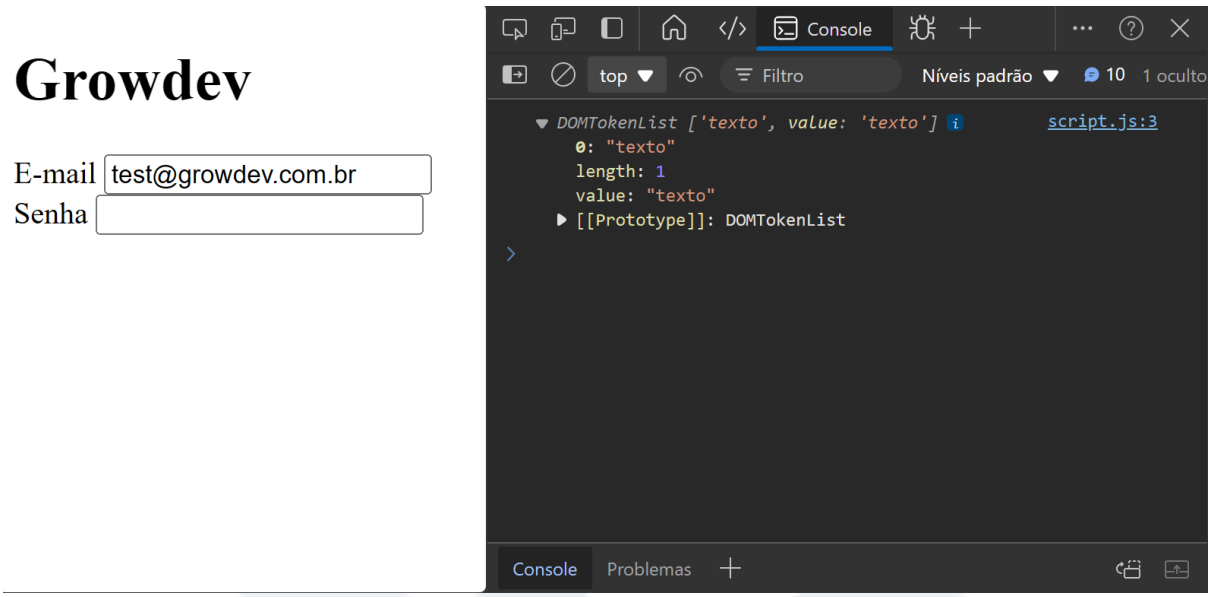
As propriedades dentro do objeto `style` podem conter **nomenclaturas diferentes** do CSS convencional. Por exemplo, a propriedade CSS `border-radius` é tratada como uma chave "`borderRadius`" no objeto. Para maiores informações sobre as propriedades CSS no DOM, consulte esta documentação.

classList

O atributo class do HTML possui uma propriedade específica dentro da interface de um `element` do DOM. A propriedade `classList` fornece o acesso a **lista de classes** que um componente possui.

```
const h1Element = document.getElementById("title");

console.log(h1Element.classList);
```



A propriedade `classList` retorna uma lista contendo cada uma das classes do elemento. O conteúdo de cada item da lista é um texto que indica o valor da classe. Ainda, `classList` possui uma propriedade chamada `value` que retorna uma string com todas as classes concatenadas e separadas por espaço em branco.

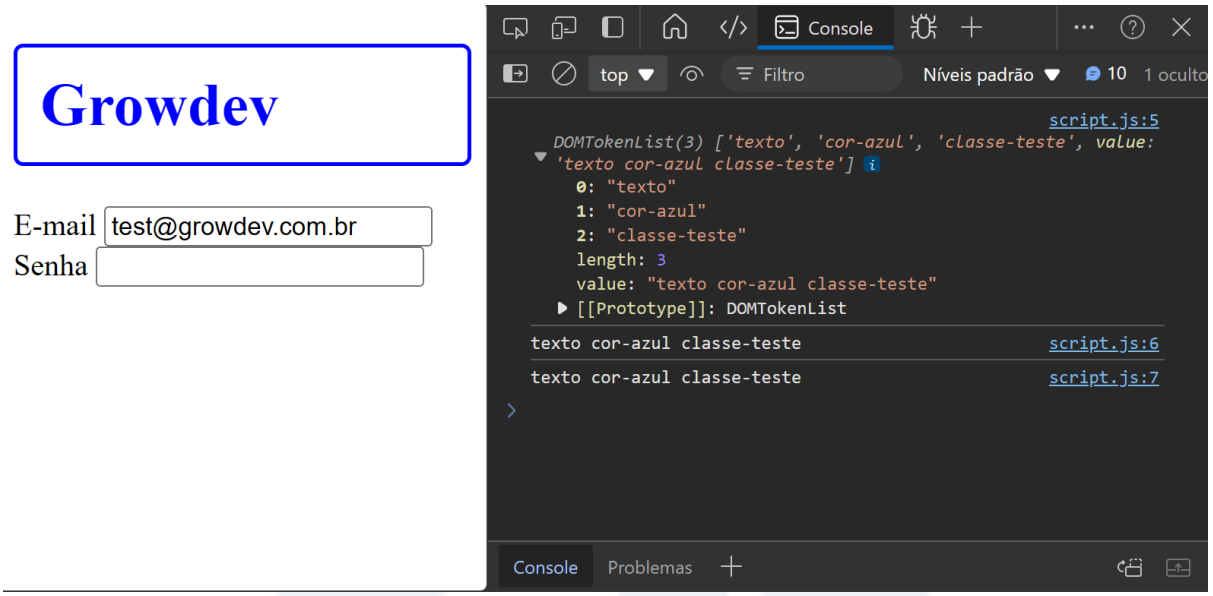
É possível ainda **adicionar** ou **remover** classes através do `classList` usando os métodos disponíveis do Javascript. A manipulação de classes via DOM é importante para mudarmos características dos elementos em tempo real, principalmente em relação ao **estilo**. Considere a seguinte regra CSS.

```
.cor-azul {
    color: blue;
    border: 2px solid blue;
    border-radius: 5px;
    padding: 12px;
}
```

Para adicionarmos uma classe através do `classList`, usamos o método add passando como parâmetro o nome da classe desejada. No exemplo abaixo, duas classes são adicionadas ao elemento `<h1>` referente ao título no HTML.

```
const h1Element = document.getElementById("title");
h1Element.classList.add("cor-azul");
h1Element.classList.add("classe-teste");

console.log(h1Element.classList);
console.log(h1Element.classList.value);
console.log(h1Element.className);
```



Com a adição de duas novas classes, o `classList` retorna então uma lista com 3 elementos. Seu `value` passa a conter uma string com a união destas três classes. Além do `value` do `classList`, outra forma de acessar o nome final da classe do elemento é a propriedade `className` presente na interface do próprio `element`.

Observe que a adição da classe “cor-azul” modificou a página para que o elemento alvo recebesse o estilo definido pela regra CSS. Logo a estilização da página pode ser facilmente e dinamicamente alterada através do DOM.

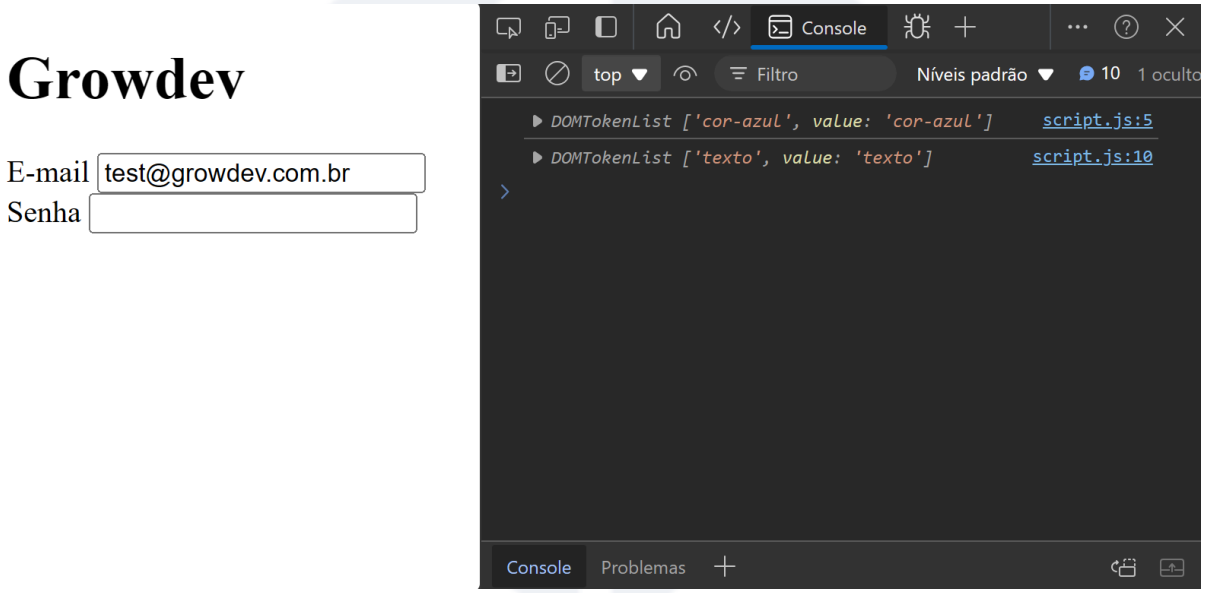
A remoção de uma `class` também é possível através do método remove do `classList`. O parâmetro de entrada é o nome da classe a ser removida. Outro método útil é o `toggle`. Com este último, o comportamento é de adição quando a classe não existe e de remoção quando ela existe no elemento alvo.

```
const h1Element = document.getElementById("title");
h1Element.classList.toggle("cor-azul");
h1Element.classList.remove("texto");

console.log(h1Element.classList);

h1Element.classList.toggle("texto");
h1Element.classList.toggle("cor-azul");

console.log(h1Element.classList);
```



O exemplo acima faz um `toggle` na `class` “cor-azul” do elemento <h1>, que ainda não possui a classe; portanto ela é adicionada ao `classList`. Por outro lado, a classe “texto” é removida com o comando `remove`. Assim, a primeira linha do console mostra apenas “cor-azul” como classe do elemento.

Logo em seguida, ambas as classes “texto” e “cor-azul” são submetidas ao método `toggle`. Como “cor-azul” era uma classe existente, ela foi removida. Já “texto” havia sido removida anteriormente, portanto o `toggle`

a adicionou novamente ao `classList`. Por isso, a última linha do console mostra apenas “texto” como classe do elemento.

REFERÊNCIAS

1. O que é DOM (Document Object Model)? Trabalhando com DOM em JavaScript
2. Introdução ao DOM - APIs da Web | MDN
3. JavaScript DOM Tutorial
4. HTML DOM Style object
5. Document - Web APIs | MDN

BONS ESTUDOS

growdev