SENAI Serviço Nacional de Aprendizagem Industrial

Manipulação de Propriedades de Texto com JavaScript

Introdução

O JavaScript atua como a "inteligência" por trás da página, permitindo que você altere o conteúdo, o estilo e até mesmo a estrutura de um documento HTML em tempo real, em resposta a ações do usuário ou a eventos programados. Nosso foco será nas propriedades de texto, que são cruciais para criar interfaces de usuário dinâmicas e envolventes.

Pré-requisitos

Para aproveitar ao máximo esta apostila, é fundamental que você tenha um bom entendimento dos seguintes conceitos:

- **HTML:** Estrutura básica de documentos, elementos, atributos e semântica.
- **CSS:** Seletores, propriedades de estilo (especialmente as relacionadas a texto como color, font-size, text-align, etc.), e como aplicar estilos a elementos HTML.
- **JavaScript Básico:** Variáveis, tipos de dados, operadores, estruturas de controle (condicionais e laços), funções e o conceito de DOM (Document Object Model) em um nível introdutório.

O que é o DOM e por que ele é importante para manipulação de texto?

O **Document Object Model (DOM)** é uma interface de programação para documentos HTML e XML. Ele representa a estrutura de um documento como uma árvore de objetos, onde cada nó da árvore corresponde a uma parte do documento (um elemento, um atributo, um texto, etc.). O JavaScript utiliza o DOM para acessar e manipular esses elementos, permitindo que alteremos dinamicamente o conteúdo e o estilo de uma página web [6].

Quando falamos em manipular propriedades de texto com JavaScript, estamos essencialmente interagindo com o DOM para:

1. **Acessar** elementos HTML que contêm texto.
2. **Modificar** o conteúdo textual desses elementos.
3. **Alterar** os estilos CSS aplicados a esses textos.

Propriedades JavaScript para Manipulação de Conteúdo de Texto

Existem algumas propriedades chave em JavaScript que nos permitem acessar e modificar o conteúdo textual de elementos HTML. As mais importantes são textContent e innerHTML.

1. textContent

A propriedade textContent define ou retorna o conteúdo de texto de um nó e todos os seus descendentes. Ela é útil quando você precisa obter ou definir apenas o texto visível de um elemento, ignorando qualquer marcação HTML interna [1, 2].

Características:

- Retorna o texto puro, sem tags HTML.
- É mais seguro para definir conteúdo fornecido pelo usuário, pois evita a injeção de HTML malicioso (cross-site scripting - XSS).
- Geralmente tem melhor performance do que innerHTML para manipulação de texto puro.

Exemplo:

```
<p id="meuParagrafo">Este é um <span>parágrafo</span> de exemplo.</p>
<button onclick="mudarTexto()">Mudar Texto</button>
<script>
   function mudarTexto() {
      const paragrafo = document.getElementById('meuParagrafo');
      console.log(paragrafo.textContent); // Saída: "Este é um parágrafo de exemplo."
      paragrafo.textContent = 'O novo texto do parágrafo.';
   }
</script>
```

2. innerHTML

A propriedade innerHTML define ou retorna o conteúdo HTML (incluindo tags) de um elemento. Ela é a maneira mais fácil de modificar o conteúdo de um elemento HTML, permitindo que você insira não apenas texto, mas também outras tags HTML [3].

Características:

- Retorna ou define o conteúdo HTML completo de um elemento, incluindo tags e texto.
- Permite a inserção de HTML dinâmico.

- Deve ser usada com cautela ao lidar com conteúdo fornecido pelo usuário, pois pode abrir portas para ataques XSS se não for sanitizada adequadamente.

Exemplo:

```
<div id="minhaDiv">Olá, <b>mundo</b>!</div>
<button onclick="mudarHTML()">Mudar HTML</button>
<script>
   function mudarHTML() {
      const minhaDiv = document.getElementById('minhaDiv');
      console.log(minhaDiv.innerHTML); // Saída: "Olá, <b>mundo</b>!"
      minhaDiv.innerHTML = '<h2>Novo Título</h2><p>Este é um novo conteúdo.</p>';
   }
</script>
```

Comparativo: textContent vs innerHTML

Característica	**textContent**	**innerHTML**
Conteúdo	Apenas texto puro	HTML completo (tags e texto)
Segurança	Mais seguro (evita XSS)	Menos seguro (risco de XSS se não sanitizado)
Performance	Geralmente melhor para texto puro	Pode ser mais lento devido à renderização do HTML
Uso Comum	Obter/definir texto simples	Inserir/modificar estrutura HTML complexa

Manipulação de Estilos CSS de Texto com JavaScript

Além de mudar o conteúdo, o JavaScript também pode alterar diretamente as propriedades de estilo CSS de um elemento. Isso é feito através da propriedade style de um elemento DOM.

Acessando e Modificando Estilos Inline

Cada elemento HTML no DOM possui uma propriedade style, que representa os estilos inline aplicados a esse elemento. As propriedades CSS são acessadas usando a notação camelCase em JavaScript (ex: font-size se torna fontSize, background-color se torna backgroundColor).

Exemplo:

```
<p id="estiloParagrafo">Este texto terá seu estilo alterado.</p>
<button onclick="mudarEstilo()">Mudar Estilo</button>
<script>
   function mudarEstilo() {
      const paragrafo = document.getElementById('estiloParagrafo');
      paragrafo.style.color = 'blue';
      paragrafo.style.fontSize = '20px';
      paragrafo.style.fontWeight = 'bold';
      paragrafo.style.textAlign = 'center';
   }
</script>
```

Manipulando Classes CSS

Uma abordagem mais robusta e recomendada para alterar estilos é manipular as classes CSS de um elemento. Em vez de alterar propriedades de estilo individuais, você adiciona ou remove classes que já possuem conjuntos de estilos definidos no seu arquivo CSS. Isso mantém o CSS separado do JavaScript e facilita a manutenção.

Para manipular classes, usamos a propriedade classList de um elemento, que oferece métodos como add(), remove(), toggle() e contains().

Exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Manipulação de Classes</title>
   <style>
      .texto-normal {
         color: black;
         font-size: 16px;
         font-family: Arial, sans-serif;
      }
      .texto-destaque {
         color: red;
         font-size: 24px;
         font-weight: bold;
         text-decoration: underline;
      }
```

```
    </style>
</head>
<body>
    <p id="meuTexto" class="texto-normal">Este é um texto de exemplo.</p>
    <button onclick="alternarDestaque()">Alternar Destaque</button>
    <script>
        function alternarDestaque() {
            const texto = document.getElementById('meuTexto');
            texto.classList.toggle('texto-destaque');
        }
    </script>
</body>
</html>
```

Comparativo: Estilos Inline vs. Classes CSS

Característica	**Estilos Inline (element.style.property)**	**Classes CSS (element.classList)**
Separação de Preocupações	Mistura lógica (JS) e apresentação (CSS)	Separação clara (JS para lógica, CSS para estilo)
Manutenção	Mais difícil de manter e escalar	Mais fácil de manter e reutilizar estilos
Prioridade CSS	Alta (estilos inline têm alta especificidade)	Depende da especificidade do seletor da classe
Recomendado para	Mudanças de estilo pontuais e dinâmicas	Mudanças de estado e temas visuais complexos

Métodos de String Úteis para Manipulação de Texto

Antes de inserir texto em elementos HTML, muitas vezes é necessário manipulá-lo. JavaScript oferece uma rica API de métodos de string que são extremamente úteis para preparar o conteúdo textual.

1. toUpperCase() e toLowerCase()

Convertem uma string para letras maiúsculas ou minúsculas, respectivamente.

Exemplo:

```
let nome = "Maria";
console.log(nome.toUpperCase()); // Saída: "MARIA"
let frase = "BEM-VINDO AO CURSO";
console.log(frase.toLowerCase()); // Saída: "bem-vindo ao curso"
```

2. trim()

Remove espaços em branco (espaços, tabulações, novas linhas) do início e do fim de uma string [14].

Exemplo:

```
let textoComEspacos = "   Olá, mundo!   ";
console.log(textoComEspacos.trim()); // Saída: "Olá, mundo!"
```

3. substring() e slice()

Extraem uma parte de uma string. substring() e slice() são muito semelhantes, mas slice() pode aceitar índices negativos.

Exemplo:

```
let saudacao = "Bem-vindo ao JavaScript";
console.log(saudacao.substring(0, 9)); // Saída: "Bem-vindo"
console.log(saudacao.slice(10, 12));    // Saída: "ao"
console.log(saudacao.slice(-10));      // Saída: "JavaScript"
```

4. replace() e replaceAll()

Substituem uma parte da string por outra. replace() substitui apenas a primeira ocorrência, enquanto replaceAll() (mais recente) substitui todas as ocorrências.

Exemplo:

```
let mensagem = "Olá mundo, mundo JavaScript!";
console.log(mensagem.replace("mundo", "pessoal")); // Saída: "Olá pessoal, mundo JavaScript!"
console.log(mensagem.replaceAll("mundo", "pessoal")); // Saída: "Olá pessoal, pessoal JavaScript!"
```

5. split()

Divide uma string em um array de substrings, com base em um separador especificado.

Exemplo:

```
let tags = "html,css,javascript";
let arrayTags = tags.split(',');
console.log(arrayTags); // Saída: ["html", "css", "javascript"]
```

6. concat()

Combina duas ou mais strings. O operador + ou template literals (`) são geralmente preferidos por serem mais concisos.

Exemplo:

```
let parte1 = "Front-";
let parte2 = "End";
console.log(parte1.concat(parte2)); // Saída: "Front-End"
console.log(`${parte1}${parte2}`); // Saída: "Front-End"
```