DOM

DOM (Document Object Model) é uma interface de programação que permite que scripts (como JavaScript) interajam com o conteúdo de uma página web. Ele representa a estrutura de um documento HTML ou XML como uma árvore de nós, onde cada nó pode ser um elemento, um atributo ou até um texto dentro do documento.

Em outras palavras, o DOM é uma representação em memória de uma página web, e o JavaScript pode manipulá-lo para alterar o conteúdo, a estrutura ou o estilo de uma página dinamicamente.

Como o DOM funciona:

- Estrutura de Árvore: O DOM organiza o conteúdo de uma página web como uma árvore hierárquica. Cada elemento HTML (como , <p>, <h1>, etc.) é um nó dentro dessa árvore.
- Cada unidade da árvore DOM é um nó. Existem diferentes tipos de nós, como:
    - Elemento: Representa as tags HTML (por exemplo, , <p>, <a>).
    - Texto: Representa o conteúdo textual dentro de um elemento.
    - Atributo: Representa os atributos de uma tag HTML, como class, id, href, etc.
- Navegação: Através do DOM, você pode acessar, modificar ou excluir qualquer um desses nós (elementos, texto, atributos) e até criar novos nós.

Os 6 DOMs

O DOM pode ser representado de diferentes maneiras para atender a diferentes necessidades e contextos. Embora o número "6" não seja uma definição universalmente aceita para todos os tipos de DOMs, geralmente se refere a tipos específicos de implementações ou categorias de DOM que existem em ambientes de desenvolvimento web.

Aqui está uma explicação dos **6 tipos de DOM** mais comumente discutidos no contexto de desenvolvimento web:

1. DOM Nativo (Native DOM)

O **DOM Nativo** é a implementação básica e padrão do DOM, que pode ser acessada diretamente pelos navegadores. Quando um navegador carrega uma página HTML, ele cria um **DOM Nativo** para representar o conteúdo da página. Esse modelo é a interface que o JavaScript usa para interagir com a página da web.

- **Exemplo**: Quando você usa o `document.getElementById()` ou `document.querySelector()` para acessar elementos HTML, você está interagindo com o DOM Nativo.

2. DOM de Núcleo (Core DOM)

O **DOM de Núcleo** é uma versão mais geral e independente de plataforma do DOM. Ele fornece uma interface básica para manipular documentos, como documentos XML ou qualquer outro tipo de documento, e não é dependente de qualquer linguagem de marcação específica, como HTML.

- **Exemplo**: O **DOM de Núcleo** pode ser utilizado para manipulação de documentos XML, permitindo que as linguagens de programação acessem e modifiquem os dados desses documentos de maneira programática.

3. DOM XML (XML DOM)

O **DOM XML** é uma versão do **DOM de Núcleo** com foco em documentos XML. Ele permite que você acesse e manipule documentos XML de maneira hierárquica. O **DOM XML** é muito utilizado quando se trabalha com serviços web e APIs que retornam dados no formato XML.

- **Exemplo**: Em uma aplicação que consome dados de uma API que retorna um XML, você pode usar o **DOM XML** para acessar os elementos e atributos desse XML.

4. DOM HTML (HTML DOM)

O **DOM HTML** é uma implementação do DOM projetada especificamente para documentos HTML. Ele estende o **DOM de Núcleo** com funcionalidades específicas para trabalhar com a estrutura de documentos HTML, permitindo a manipulação de elementos HTML como `<div>`, `<p>`, `<a>`, etc.

- **Exemplo**: Usar JavaScript para acessar e modificar elementos HTML como `<h1>`, `<input>`, ou adicionar classes e atributos a esses elementos. O **DOM HTML** é amplamente utilizado em desenvolvimento web.

5. DOM do Navegador (Browser DOM)

O **DOM do Navegador** é a implementação do DOM em um ambiente de navegador. Quando você usa JavaScript no navegador para manipular o DOM, você está interagindo com o **DOM do Navegador**. Ele é responsável por fornecer a interface entre o código JavaScript e a árvore de objetos que representa o conteúdo de uma página web.

- **Exemplo**: No navegador, ao usar `document.getElementById()` ou adicionar eventos a elementos HTML com `addEventListener()`, você está interagindo com o **DOM do Navegador**.

6. Virtual DOM

O **Virtual DOM** não é parte do DOM tradicional, mas é um conceito utilizado por algumas bibliotecas e frameworks de JavaScript, como o **React**. O **Virtual DOM** é uma representação em memória da árvore de elementos HTML, que permite otimizar as atualizações da interface do usuário.

- **Como funciona**: Quando há uma mudança no estado de um componente, o **Virtual DOM** é atualizado primeiro, e depois as mudanças são comparadas com o DOM real (por meio de um processo chamado **reconciliação**). Isso minimiza a quantidade de manipulação do DOM real, melhorando o desempenho.

- **Exemplo**: Quando você altera o estado de um componente em **React**, o Virtual DOM é atualizado primeiro. Em seguida, ele calcula a diferença entre o Virtual DOM e o DOM real e aplica apenas as mudanças necessárias no navegador.

Resumo dos 6 tipos de DOM:

1. **DOM Nativo**: Implementação padrão que interage diretamente com a árvore DOM no navegador.
2. **DOM de Núcleo**: Implementação geral para qualquer tipo de documento, como XML.
3. **DOM XML**: Versão do DOM para manipulação de documentos XML.
4. **DOM HTML**: Implementação específica para documentos HTML.
5. **DOM do Navegador**: A implementação do DOM que os navegadores utilizam para interagir com páginas web.
6. **Virtual DOM**: Usado em frameworks como o **React**, é uma representação em memória do DOM que otimiza a atualização da interface do usuário.

Esses **6 DOMs** representam diferentes maneiras de interagir com documentos em várias plataformas e contextos. O uso do Virtual DOM, por exemplo, permite que aplicações como o **React** façam atualizações de UI de forma eficiente sem manipular diretamente o DOM do navegador, melhorando a performance.

Manipulação do DOM com JavaScript:

JavaScript oferece vários métodos para interagir com o DOM.

Aqui estão alguns exemplos:

1. **Selecionando elementos:**
    - document.getElementById("id"): Seleciona um elemento pelo seu ID.
    - document.getElementsByClassName("classe"): Seleciona todos os elementos com uma determinada classe.
    - document.querySelector("seletor"): Seleciona o primeiro elemento que corresponde a um seletor CSS.
    - document.querySelectorAll("seletor"): Seleciona todos os elementos que correspondem ao seletor CSS.
2. **Modificando elementos:**
    - element.innerHTML: Modifica o conteúdo HTML de um elemento.
    - element.style: Modifica o estilo CSS de um elemento diretamente.
    - element.setAttribute("atributo", "valor"): Modifica um atributo de um elemento.
3. **Criando e removendo elementos:**
    - document.createElement("tag"): Cria um novo elemento HTML.
    - element.appendChild(novoElemento): Adiciona um novo elemento ao final de um outro.
    - element.remove(): Remove um elemento do DOM.
4. **Eventos: O DOM permite que você adicione eventos (como cliques de mouse, pressionamento de teclas, etc.) aos elementos. Por exemplo:**
    - element.addEventListener("click", function() {...}): Adiciona um evento de clique a um elemento.

Exemplo prático:

Imagine que você tem o seguinte HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Exemplo DOM</title>
</head>
<body>
  <h1 id="titulo">Olá, Mundo!</h1>
           <button     id="alterarTexto">Alterar
Texto</button>

  <script src="script.js"></script>
</body>
</html>
```

E no seu arquivo script.js você tem o seguinte código JavaScript:

```
// Seleciona o botão e o título
const                  botao                  =
document.getElementById('alterarTexto');
const                  titulo                 =
document.getElementById('titulo');

// Adiciona um evento ao botão
botao.addEventListener('click', function() {
```

```
 // Altera o conteúdo do título quando o botão é
clicado
 titulo.innerHTML = 'Texto Alterado!';
});
```

Neste exemplo:

- O DOM é acessado para encontrar o botão (botao) e o título (titulo).
- Um evento de clique é adicionado ao botão.
- Quando o botão é clicado, o conteúdo do título é alterado dinamicamente através do DOM.

Aqui está um exemplo prático de uma aplicação usando o DOM para criar uma lista interativa. Vamos fazer um simples gerenciador de tarefas, onde você pode adicionar, marcar como concluída e remover tarefas dinamicamente na página.

Exemplo: Gerenciador de Tarefas (To-Do List)

HTML (index.html):

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
                        <meta      name="viewport"
content="width=device-width, initial-scale=1.0">
    <title>Gerenciador de Tarefas</title>
```

```
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            padding: 10px;
        }
        #tarefaInput {
            padding: 10px;
            font-size: 16px;
            margin-right: 10px;
        }
        button {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
        }
        ul {
            list-style-type: none;
            padding-left: 0;
        }
        li {
            margin: 10px 0;
        }
        .concluida {
            text-decoration: line-through;
            color: gray;
        }
    </style>
</head>
```

```
<body>
    <h1>Gerenciador de Tarefas</h1>
          <input   type="text"   id="tarefaInput"
placeholder="Digite uma tarefa">
         <button   id="adicionarTarefa">Adicionar
Tarefa</button>

    <ul id="listaTarefas"></ul>

    <script src="script.js"></script>
</body>
</html>
```

JavaScript (script.js):

```
// Seleciona os elementos do DOM
const                inputTarefa                =
document.getElementById('tarefaInput');
const              botaoAdicionar               =
document.getElementById('adicionarTarefa');
const                listaTarefas               =
document.getElementById('listaTarefas');

// Função para adicionar tarefa
function adicionarTarefa() {
                     const     textoTarefa     =
inputTarefa.value.trim();

    if (textoTarefa === "") {
```

```
        alert("Por favor, insira uma tarefa!");
        return;
    }

    // Cria um novo item de lista (li)
    const li = document.createElement('li');

    // Cria um texto dentro do item da lista
                        const      texto      =
document.createTextNode(textoTarefa);
    li.appendChild(texto);

    // Cria um botão de "Concluir" para a tarefa
                    const     botaoConcluir     =
document.createElement('button');
    botaoConcluir.textContent = "Concluir";
    botaoConcluir.style.marginLeft = "10px";
    botaoConcluir.onclick = function() {
        li.classList.toggle('concluida');
    };

    // Cria um botão de "Remover" para a tarefa
                    const     botaoRemover     =
document.createElement('button');
    botaoRemover.textContent = "Remover";
    botaoRemover.style.marginLeft = "10px";
    botaoRemover.onclick = function() {
        listaTarefas.removeChild(li);
    };
```

```
        // Adiciona os botões ao item da lista
        li.appendChild(botaoConcluir);
        li.appendChild(botaoRemover);

          // Adiciona o item de lista à lista de
    tarefas
        listaTarefas.appendChild(li);

        // Limpa o campo de input
        inputTarefa.value = "";
    }

    // Adiciona a tarefa quando o botão é clicado
    botaoAdicionar.addEventListener('click',
    adicionarTarefa);

    // Permite adicionar tarefa pressionando a tecla
    "Enter"
    inputTarefa.addEventListener('keypress',
    function(event) {
        if (event.key === 'Enter') {
            adicionarTarefa();
        }
    });
```

Como funciona a aplicação:

1. **Estrutura HTML**:

- Um campo de entrada (<input>) permite ao usuário digitar uma tarefa.
    - Um botão (<button>) para adicionar a tarefa à lista.
    - Uma lista não ordenada (<ul>) que irá conter as tarefas.

2. **JavaScript**:

    - Quando o botão "Adicionar Tarefa" é clicado ou quando a tecla "Enter" é pressionada no campo de input, a função adicionarTarefa() é executada.
    - Essa função cria um novo item de lista (<li>) com o texto da tarefa inserido, e dois botões: um para marcar a tarefa como concluída e outro para remover a tarefa.
    - O botão de "Concluir" adiciona ou remove a classe concluida, que aplica o estilo de "riscado" na tarefa.
    - O botão de "Remover" deleta a tarefa da lista.

Demonstração:

- Ao digitar uma tarefa e clicar em "Adicionar Tarefa", a tarefa aparece na lista com os botões "Concluir" e "Remover".
- Clicar em "Concluir" marca a tarefa como concluída (com um texto riscado).
- Clicar em "Remover" exclui a tarefa da lista.

Essa aplicação é uma forma simples de demonstrar como o DOM pode ser manipulado para criar funcionalidades interativas em uma página web.

Conclusão

O DOM é uma interface de programação que permite que os scripts modifiquem e interajam com as páginas, o DOM possui 6 diferentes tipos específicos, mas não são universalmente aceitas pela comunidade de

desenvolvedores. É importante aprender sobre o DOM porque ele ajuda nos processos de criação, desenvolvimento, e modificação de uma página.