DOM

O **DOM** (Document Object Model) é uma interface de programação (API) que define a estrutura dos documentos HTML e XML como uma árvore de objetos. Em outras palavras, o DOM é uma representação em forma de árvore do conteúdo e estrutura de um documento web, onde cada parte do documento (como elementos, atributos e texto) é representada como um "nó" na árvore.

Como o DOM Funciona?

Quando um navegador carrega uma página HTML, ele a converte em uma estrutura de dados interna chamada "DOM", que permite que o JavaScript interaja com o conteúdo da página. A ideia principal do DOM é transformar a estrutura do HTML em objetos acessíveis e manipuláveis, o que permite que você altere o conteúdo, o estilo e até a estrutura da página dinamicamente através de JavaScript.

Estrutura do DOM

Imagine que você tem um documento HTML como o seguinte:

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo DOM</title>
  </head>
  <body>
    <h1 id="titulo">Olá, Mundo!</h1>
    <p class="paragrafo">Este é um parágrafo.</p>
    <button id="botao">Clique aqui</button>
  </body>
</html>
```

O DOM dessa página seria estruturado de forma hierárquica assim:

```
HTML (Elemento raiz)
 ├── HEAD
 │    └── TITLE
 └── BODY
      ├── H1 (id="titulo")
      ├── P (class="paragrafo")
      └── BUTTON (id="botao")
```

Cada um desses elementos (`<html>`, `<head>`, `<body>`, etc.) é um **nó** da árvore DOM. O JavaScript pode acessar esses nós para interagir com o conteúdo da página.

Como JavaScript Interage com o DOM

O JavaScript usa o DOM para acessar e modificar a estrutura e o conteúdo do documento. A manipulação do DOM no JavaScript pode ser dividida em algumas operações principais:

1. Acessar Elementos

JavaScript pode acessar os elementos do DOM usando várias funções. As mais comuns são:

- **`getElementById()`**: Acessa um elemento pelo seu `id`.
- **`getElementsByClassName()`**: Acessa todos os elementos com uma determinada classe.
- **`querySelector()`**: Acessa o primeiro elemento que corresponde a um seletor CSS.
- **`querySelectorAll()`**: Acessa todos os elementos que correspondem a um seletor CSS.

Exemplo:

```
let titulo = document.getElementById('titulo');
let paragrafo = document.querySelector('.paragrafo');
```

2. Alterar Conteúdo

Uma vez que um elemento é acessado, podemos alterar seu conteúdo ou propriedades. Por exemplo:

- **`innerHTML`**: Modifica o conteúdo HTML de um elemento.
- **`innerText`** ou **`textContent`**: Modifica o conteúdo de texto de um elemento.

Exemplo:

```
titulo.innerText = "Novo Título!";
paragrafo.innerHTML = "<strong>Texto em negrito!</strong>";
```

3. Alterar Estilos

O JavaScript pode alterar diretamente os estilos de um elemento com a propriedade `style`:

```
titulo.style.color = "blue";
titulo.style.fontSize = "30px";
```

4. Adicionar ou Remover Elementos

Você pode criar novos elementos e inseri-los na página, ou remover elementos existentes:

- `createElement()`: Cria um novo elemento.
- `appendChild()`: Adiciona um novo elemento como filho de um outro elemento.
- `removeChild()`: Remove um elemento de seu elemento pai.
-

Exemplo:

```
let novoParagrafo = document.createElement('p');
novoParagrafo.innerText = "Este é um novo parágrafo.";
document.body.appendChild(novoParagrafo);
```

5. Eventos

O DOM permite adicionar interatividade à página web através de **eventos**. Por exemplo, você pode adicionar um evento de clique em um botão:

```
let botao = document.getElementById('botao');
botao.addEventListener('click', function() {
  alert("Você clicou no botão!");
});
```

Exemplo Prático de Manipulação do DOM

Aqui está um exemplo simples de uma página HTML com um script JavaScript que manipula o DOM:

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Exemplo DOM</title>
</head>
<body>
  <h1 id="titulo">Bem-vindo ao DOM!</h1>
  <button id="botao">Clique para mudar o título</button>

  <script>
    // Acessando os elementos
    const titulo = document.getElementById('titulo');
    const botao = document.getElementById('botao');

    // Alterando o conteúdo do título
    titulo.innerText = "Alterado pelo JavaScript!";

    // Adicionando um evento ao botão
    botao.addEventListener('click', function() {
      titulo.innerText = "Você clicou no botão!";
    });
  </script>
</body>
</html>
```

Neste exemplo:

1. O título inicial da página é "Bem-vindo ao DOM!".
2. Quando o botão é clicado, o título é alterado para "Você clicou no botão!".

Os 6 tipos de DOM

Os **6 DOMs** a que você se refere provavelmente são os tipos de representações ou implementações do **Document Object Model (DOM)**. 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.

Aplicação

Ao manipular o DOM, você pode criar interações complexas e dinâmicas em páginas web, sem a necessidade de recarregar a página, oferecendo uma experiência mais fluida e interativa para o usuário.

Uma aplicação comum do DOM é criar uma página interativa onde o conteúdo pode ser alterado em resposta às ações do usuário, sem que seja necessário recarregar a página. Vou mostrar um exemplo prático de como o DOM pode ser usado para criar uma **lista interativa**, onde o usuário pode adicionar e remover itens da lista dinamicamente.

Aplicação: Lista de Tarefas Interativa

Este exemplo cria uma lista de tarefas simples. O usuário pode adicionar tarefas à lista e removê-las clicando em um botão.

HTML e JavaScript:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Lista de Tarefas</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    #tarefaInput {
      padding: 10px;
      margin-right: 10px;
      width: 200px;
    }
    button {
      padding: 10px;
      cursor: pointer;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      margin: 10px 0;
```

```
      display: flex;
      align-items: center;
    }
    .remover {
      margin-left: 10px;
      color: red;
      cursor: pointer;
    }
  </style>
</head>
<body>

  <h1>Lista de Tarefas</h1>

  <input type="text" id="tarefaInput" placeholder="Digite uma tarefa">
  <button id="adicionarBtn">Adicionar Tarefa</button>

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

  <script>
    // Acessando os elementos do DOM
    const inputTarefa = document.getElementById('tarefaInput');
    const botaoAdicionar = document.getElementById('adicionarBtn');
    const listaTarefas = document.getElementById('listaTarefas');

    // Função para adicionar tarefa
    botaoAdicionar.addEventListener('click', function() {
      const tarefaTexto = inputTarefa.value.trim();

      if (tarefaTexto !== "") {
        // Criando o item de lista
        const li = document.createElement('li');

        // Criando o texto da tarefa
        const textoTarefa = document.createTextNode(tarefaTexto);
        li.appendChild(textoTarefa);

        // Criando o botão de remoção
        const btnRemover = document.createElement('span');
        btnRemover.textContent = "Remover";
        btnRemover.classList.add('remover');

        // Adicionando o evento de remoção
        btnRemover.addEventListener('click', function() {
          listaTarefas.removeChild(li);
        });

        li.appendChild(btnRemover);
        listaTarefas.appendChild(li);

        // Limpando o campo de input
```

```
        inputTarefa.value = "";
      } else {
        alert("Por favor, insira uma tarefa.");
      }
    });

    // Opcional: permite adicionar tarefa pressionando Enter
    inputTarefa.addEventListener('keypress', function(event) {
      if (event.key === 'Enter') {
        botaoAdicionar.click();
      }
    });
  </script>

</body>
</html>
```

Como Funciona:

1. **HTML**:
    - Um campo de input (`<input>`) para o usuário digitar uma tarefa.
    - Um botão (`<button>`) para adicionar a tarefa à lista.
    - Uma lista não ordenada (`<ul>`) onde as tarefas serão listadas.
2. **JavaScript**:
    - O **DOM** é manipulado para criar novos itens de lista dinamicamente.
    - O evento `click` no botão de adicionar (`adicionarBtn`) chama uma função que:
        1. Lê o valor digitado no campo de input.
        2. Cria um novo elemento `<li>` para representar a tarefa.
        3. Adiciona a tarefa à lista (`ul`).
        4. Adiciona um botão de **remover** para excluir a tarefa da lista.
    - Quando o botão **remover** é clicado, a tarefa é removida da lista.
    - O usuário pode também pressionar **Enter** para adicionar a tarefa (por conveniência).
3. **Estilos CSS**:
    - A página tem um estilo simples para deixar a lista mais legível e as interações mais claras, com destaque para o botão de remover.

Funcionalidades:

- **Adicionar tarefas**: O usuário digita a tarefa no campo de texto e clica no botão "Adicionar Tarefa" para que a tarefa apareça na lista.
- **Remover tarefas**: Ao lado de cada tarefa há um botão "Remover". Quando o usuário clica neste botão, a tarefa é removida da lista.

- **Adicionar com Enter**: O usuário pode pressionar a tecla "Enter" para adicionar a tarefa, o que torna a interação mais rápida.

Como isso usa o DOM:

- **Criar elementos**: O JavaScript usa `createElement()` para criar os elementos `<li>` e o botão de remoção.
- **Manipular conteúdo**: O `innerText` é usado para definir o texto das tarefas e do botão de remoção.
- **Adicionar elementos à página**: O `appendChild()` é utilizado para adicionar os novos itens à lista existente (`ul`).
- **Eventos**: Usamos `addEventListener()` para detectar o clique no botão e a tecla pressionada (Enter) para adicionar tarefas, e também para remover tarefas da lista.

Resultado:

Essa aplicação simples permite que o usuário adicione e remova tarefas em uma lista interativa, demonstrando o poder do DOM para criar páginas web dinâmicas e responsivas sem a necessidade de recarregar a página.

conclusão

O DOM seria como uma árvore de nós onde o javascript onde o javascript permite manipular a estrutura e o conteúdo do DOM com o uso de diversas operações e funções diversas e com funcionalidades e finalidades diferentes, também existem 6 tipos de DOM, cada um deles com sua finalidade e funcionamento diversos, O DOM serve para a manipulação e modificação mais simples, rápida e direta do código, ele torna a modificação e manipulação do código mais clara, auxiliando no desenvolvimento da página através do DOM e javascript.