**O que é o DOM?**

O Document Object Model (DOM) é uma interface de programação que representa documentos HTML e XML como objetos, permitindo que sejam manipulados via código [JavaScript](https://www.escoladnc.com.br/blog/javascript/" \t "_blank). Ele é essencial para a criação de páginas web interativas e dinâmicas, onde o conteúdo pode ser alterado sem a necessidade de recarregar a página.

* O DOM é uma interface de programação que representa documentos HTML e XML como objetos
* Permite a manipulação via código [JavaScript](https://www.escoladnc.com.br/blog/javascript/" \t "_blank)
* Essencial para criar páginas web interativas e dinâmicas

**O poder do DOM**

O DOM é representado como uma árvore, refletindo a estrutura hierárquica do HTML. No topo está o objeto ‘window’, seguido do objeto ‘document’ que representa o documento HTML. Dentro do ‘document’ estão os elementos , e , cada um com seus elementos internos. O poder do DOM está em poder acessar e manipular qualquer parte dessa árvore via JavaScript.

* O DOM é representado como uma árvore, refletindo a estrutura hierárquica do HTML
* Permite acessar e manipular qualquer parte da árvore via [JavaScript](https://www.escoladnc.com.br/blog/javascript/" \t "_blank)

**Selecionando elementos do DOM**

Para manipular algum elemento, primeiro precisamos selecioná-lo. Existem várias formas de fazer isso com o DOM:

* getElementById: seleciona elemento pelo id
* getElementsByClassName: seleciona elementos por classe
* getElementsByTagName: seleciona elementos por tag
* querySelector: seleciona o primeiro elemento que corresponde a um seletor CSS especificado
* querySelectorAll: seleciona todos os elementos que correspondem a um seletor CSS especificado

**Selecionando elementos**

Ao desenvolver aplicações web, é fundamental compreender as diferentes maneiras de selecionar elementos [HTML](https://www.escoladnc.com.br/blog/guia-de-html-css-e-javascript-para-iniciantes/)utilizando JavaScript. Existem várias formas de realizar essa seleção, e cada uma possui suas características específicas.

* O método getElementById permite selecionar um elemento com base no seu ID, retornando apenas um elemento.
* getElementsByTagName seleciona elementos com base em sua tag name, retornando um array com os elementos correspondentes.
* getElementsByClassName seleciona elementos com base em sua classe CSS, também retornando um array com os elementos correspondentes.
* O método querySelector retorna o primeiro elemento correspondente ao seletor CSS passado, enquanto o querySelectorAll retorna todos os elementos correspondentes em um array. Ambos aceitam qualquer seletor CSS válido.

**Manipulando elementos**

Além de selecionar elementos, também é possível manipulá-los de diversas formas utilizando JavaScript. Essa manipulação pode incluir a alteração de estilos, textos e atributos dos elementos.

* Para alterar estilos de CSS, podemos utilizar a propriedade .style, permitindo a modificação de propriedades como cor, tamanho de fonte, entre outras.
* A propriedade .textContent é utilizada para alterar o texto de um elemento de forma direta.
* Acessar e alterar atributos de elementos é possível utilizando os métodos .getAttribute() e .setAttribute().

**Adicionando e removendo elementos**

Ao trabalhar com o Document Object Model (DOM) em JavaScript, é essencial compreender como adicionar e remover elementos. Existem diferentes métodos para realizar essas ações, como o uso de .innerHTML e .append().

* Utilize .innerHTML e .append() para adicionar novos elementos ao DOM
* A utilização de .innerHTML e .append() proporciona flexibilidade na criação e adição de elementos
* Entenda a diferença entre a adição de elementos utilizando .innerHTML e .append()

**Boas práticas ao manipular o DOM**

Ao manipular o DOM, é fundamental seguir boas práticas para garantir a eficiência e a organização do código. Essas práticas incluem o cache de elementos, a minimização do reflow do navegador e o cuidado com vazamentos de memória ao remover elementos.

* Realize o cache de elementos que serão reutilizados para melhorar a performance
* Evite manipulações no carregamento da página e aguarde o evento DOMContentLoaded
* Minimize o reflow do navegador ao realizar alterações nos estilos
* Atente-se aos vazamentos de memória ao remover elementos e event listeners associados
* Utilize fragmentos de documento ao adicionar vários elementos de uma só vez.

## Performance e o DOM

### Otimização da Manipulação do DOM

A manipulação ineficiente do DOM pode levar a problemas de desempenho, especialmente em aplicações grandes.

#### Dicas de Otimização

* **Minimize as Alterações no DOM:** Alterações frequentes no DOM podem ser custosas. Agrupe mudanças e aplique-as de uma vez.
* **Use CSS para Alterações Visuais:** Sempre que possível, prefira alterações de estilo via CSS em vez de JavaScript.

## Melhores Práticas

1. **Evite Excesso de Dependência do DOM:** Demasiada interação com o DOM pode tornar o aplicativo lento e difícil de manter.
2. **Acessibilidade:** Ao manipular o DOM, considere a acessibilidade, garantindo que as mudanças não prejudiquem a experiência do utilizador.
3. **Segurança:** Tenha cuidado ao usar **innerHTML** para evitar ataques de Cross-Site Scripting (XSS).

## Conclusão

O DOM é um aspecto essencial do desenvolvimento web em JavaScript, permitindo a criação de interfaces dinâmicas e interativas. Compreender a manipulação do DOM e seguir as melhores práticas é crucial para qualquer desenvolvedor web que busca criar aplicações rápidas, eficientes e acessíveis.

Manipulação de eventos (cliques, teclas, etc.).

O que são eventos?

Os eventos são basicamente um conjunto de ações que são realizadas em um determinado elemento da página web, seja ele um texto, uma imagem, ou uma div, por exemplo. Muitas das interações do usuário que está visitando sua página com o conteúdo do seu site podem ser consideradas eventos.

Existe uma infinidade de eventos definidos para uso em JavaScript, abaixo podemos ver alguns dos principais e mais utilizados:

|  |  |
| --- | --- |
| **onBlur** | remove o foco do elemento |
| **onChange** | muda o valor do elemento |
| **onClick** | o elemento é clicado pelo usuário |
| **onFocus** | o elemento é focado |
| **onKeyPress** | o usuário pressiona uma tecla sobre o elemento |
| **onLoad** | carrega o elemento por completo |
| **onMouseOver** | define ação quando o usuário passa o mouse sobre o elemento |
| **onMouseOut** | define ação quando o usuário retira o mouse sobre o elemento |
| **onSubmit** | define ação ao enviar um formulário |

### Como usar os eventos em JavaScript

Existem diversas maneiras de se aplicar esses eventos aos elementos **HTML**, são elas:

* Inline
* Em um arquivo externo, usando um manipulador de eventos

### Evento Inline

Vamos ver um exemplo de como usar um evento de maneira inline em JavaScript. Dessa forma é preciso que tudo seja definido diretamente na tag do elemento, dessa forma:

https://www.devmedia.com.br/trabalhando-com-eventos-em-javascript/2852