

PROGRAMAÇÃO FRONT END II

Fabiano Berlinck Neumann



sagah
SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

DOM: *document object model*

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Definir o *document object model* (DOM).
- Descrever a estrutura de árvore do DOM.
- Reconhecer as interfaces Core, HTML e CSS do DOM.

Introdução

A troca de dados entre computadores e dispositivos distribuídos pelo planeta está evoluindo de forma cada vez mais acelerada e, com isso, vem transformando a forma como as pessoas acessam informações *on-line* e como interagem com estas. Junto a essa evolução, há um aumento na complexidade das interfaces para que seja possível atender às necessidades dos diferentes perfis de usuários.

O desenvolvedor Web é responsável por criar essas interfaces, mas este não é seu único trabalho, especialmente se este profissional for um desenvolvedor *full stack*, que trabalha tanto com *front end* quanto *back end*. No caso do último, o profissional programa a parte do sistema que irá rodar nos servidores. Já no caso do *front end*, a parte que é desenvolvida tem relação com o código que irá rodar no navegador do computador, no caso das aplicações *desktop*, ou em outros dispositivos dos usuários conhecidos como clientes na arquitetura cliente-servidor.

Como as empresas estão cada vez mais presentes na internet, o desenvolvedor *front end* tem ganhado destaque no cenário de desenvolvimento Web, tendo em vista a necessidade crescente de interfaces mais amigáveis, de fácil entendimento e rememoração, que sejam esteticamente agradáveis, divertidas e interessantes. Isso ocorre em razão da preocupação com a experiência do usuário, cada vez mais relevante para que empresas convertam usuários do topo do funil de vendas em compradores de seus produtos e serviços.

Neste capítulo, você conhecerá a relação entre algumas das tecnologias mais importantes para o desenvolvimento *front end*, como o HTML, o CSS e o JavaScript, de forma que você consiga definir o DOM e descrever a estrutura de sua árvore, além de compreender suas interfaces Core, HTML e CSS.

Definição

No início dos estudos na área de desenvolvimento Web e, em grande parte das tecnologias de desenvolvimento móvel híbridas, em que um código gera aplicativos nas diversas plataformas, é essencial conhecer a estrutura do HTML e do CSS, nas versões 5 e 3, respectivamente, que são as tecnologias que permitem informar aos navegadores e aplicativos o que mostrar e aspectos referentes à aparência dos elementos a serem apresentados.

Para Castiglioni (2018, documento *on-line*), além dos crescentes números de páginas Web disponíveis diariamente, cresce também a quantidade de técnicas, padrões e boas práticas de desenvolvimento de *software*. A semântica, conhecida como sentido, é uma dessas práticas que permite aos usuários entender o significado de cada uma das partes dos *sites* em que estão navegando. Além disso, desenvolver com uma boa semântica é fundamental para obter melhor posicionamento no *ranking* do Google, o que melhora a otimização para serviço de busca (*search engine optimization* — SEO) do *site*, e no ponto de vista da acessibilidade, além de facilitar no entendimento e na manutenção do código.

De acordo com Nascimento (2019, documento *on-line*), uma vez criadas as páginas com boa estrutura HTML e estilização, é necessário adicionar um comportamento dinâmico e interativo a elas; para tanto, uma das linguagens mais utilizadas é o JavaScript. Com ela é possível criar eventos, filtros de pesquisa e apresentação dos dados, manipular elementos, validar formulários, assim como enviar requisições para determinadas URLs e trabalhar com a resposta sem que as páginas precisem ser carregadas novamente, método conhecido como AJAX.

Diversas dessas ações em JavaScript e das interações dos usuários na página são possíveis devido a manipulação do DOM, que é, segundo Maldonado (2018, documento *on-line*), uma interface de representação de como os documentos HTML e XML são lidos pelos navegadores. Após a leitura da página pelo navegador, é criado um objeto que faz essa representação estruturada do documento e define meios para que essa estrutura possa ser acessada e manipulada.

Franklyn (2019, documento *on-line*) afirma que o DOM é a estrutura e o conteúdo de um documento Web, representados por dados dos objetos e um documento HTML e XML na memória do computador ou dispositivo. Tal documento pode ser apresentado no navegador ou como fonte HTML e, devido a sua representação, ele pode ser modificado a partir de linguagens de *script*. Ainda, o autor afirma que existem dois padrões que podem ser implementados na maioria dos navegadores mais recentes, o W3C DOM e o WHATWG DOM, e que muitos navegadores estendem esses padrões. Dessa forma, deve-se utilizá-los com cuidado, já que os documentos podem ser processados por diferentes DOMs desses vários navegadores.

O DOM e o JavaScript eram inicialmente interligados, mas, com o passar do tempo, avançaram no caminho de se tornar entidades separadas. Apesar de o JavaScript ser a linguagem mais utilizada para manipular o DOM, suas implementações podem ser realizadas em qualquer linguagem, como em Python, já que o DOM foi planejado para ser independente de uma linguagem de programação específica.

Apesar de o DOM não ser uma linguagem de programação e não ter sido planejado para ser utilizado com uma única linguagem de programação, sem ele, as linguagens, como JavaScript, não teriam modelos de páginas Web, documentos HTML e XML, nem as partes que compõem, por exemplo, os elementos da página. Para que esses elementos possam ser acessados e manipulados por meio do DOM, com uma linguagem de programação, cada um dos elementos do documento, como ele próprio, o cabeçalho, as tabelas do documento, os conteúdos das células da tabela e seu cabeçalho, faz parte do modelo de objeto do documento.

Ao alterar esse modelo, a partir de uma linguagem de programação escondida pelo desenvolvedor (como o JavaScript, no caso deste capítulo), também é alterada a página Web, o que torna mais fácil trabalhar na página com o DOM do que diretamente, via código HTML ou CSS. Tudo ocorre em razão do objeto `document`, que é o objeto que cede o acesso à árvore DOM do navegador, assunto que será tratado a seguir, no próximo tópico.

Dessa forma, tudo que for criado pelo navegador no modelo da página pode ser acessado por meio deste objeto `document`, utilizado principalmente para atualizar a página, normalmente junto ao AJAX, assim como para mover os itens dentro desta ou criar efeitos CSS, sem haver necessidade de ficar atualizando a página.

Estrutura da árvore DOM

A parte HTML do modelo DOM é construída como uma árvore de objetos, como podemos observar na Figura 1. O objeto `document` é o objeto pai de todos os outros objetos HTML da árvore e, por consequência, da página Web. Além dele, existem outros dois objetos que são filhos do objeto `window`, o `location` e o `history`, que podem ser utilizados, respectivamente, para capturar a URL atual para redirecionar o navegador para outra página e para obter as URLs visitadas pelo usuário dentro do navegador, até chegar à página em questão.

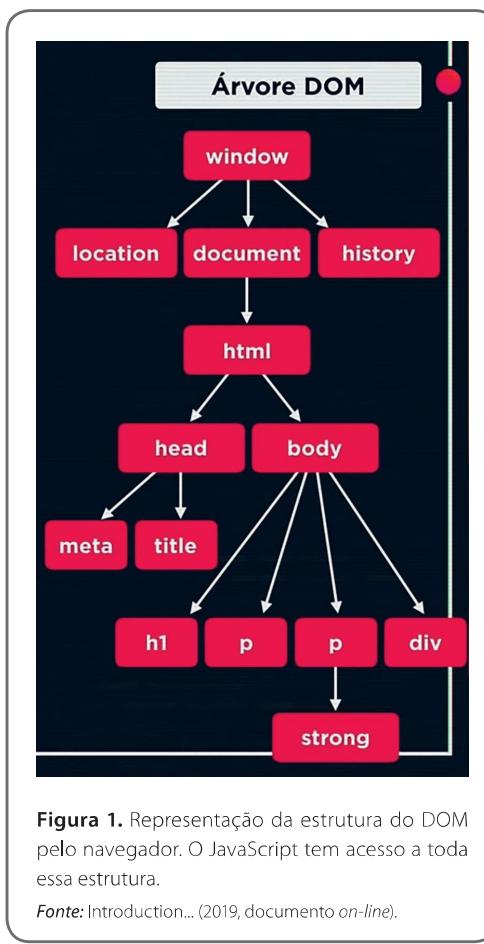


Figura 1. Representação da estrutura do DOM pelo navegador. O JavaScript tem acesso a toda essa estrutura.

Fonte: Introduction... (2019, documento on-line).

De acordo com o site W3Schools, com o modelo do objeto, a linguagem JavaScript é capaz de criar HTML de forma dinâmica, com a possibilidade de alterar todos os elementos HTML da página, todos seus estilos CSS, remover elementos e atributos HTML existentes, adicionar novos elementos e atributos HTML, assim como reagir aos eventos HTML existentes na página e criar novos eventos do mesmo tipo (JAVASCRIPT..., 2019, documento *on-line*).

Veja, no tópico a seguir, como é possível acessar os objetos da árvore do DOM.

Acessando o DOM

Ao criar um *script* que seja embutido entre as tags `<script>` e `</script>`, ou que seja incluído na página Web via instrução de carregamento de *script*, é possível utilizar os objetos `document` ou `window` para realizar a manipulação do documento da página. Com estes objetos é possível, também, obter seus filhos neste documento, ou seja, os demais elementos da página Web em questão que podem ser alterados para adicionar comportamento ao *site*.

Além disso, dependendo do que se deseja programar, trabalhar com o DOM pode ser consideravelmente simples, como no caso de exibir uma mensagem de alerta a partir da função `alert()` do objeto `window`, como no exemplo de código a seguir.

```
<body onload="window.alert('Hello DOM World!');">
```

Entretanto, também existem métodos mais sofisticados, como no caso da necessidade de criação de conteúdo novo, como é possível observar no exemplo de código a seguir.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="utf-8"/>
    <title>Título da Página na Aba</title>
    <script>
        window.onload = function() {
            var heading = document.createElement("h1");
            var heading_text = document.createTextNode("Texto do
Título");
            heading.appendChild(heading_text);
```

```
        document.body.appendChild(heading);
    }
</script>
</head>
<body>
</body>
</html>
```

Nesse caso, a função será chamada quando o documento for carregado, criando um novo elemento de título `h1`, adicionando o texto a esse elemento e, por fim, acrescentando o `h1` no final do código que está entre as tags `<body>` e `</body>`, que, no caso, estava vazia antes da página ser carregada.

Além dos métodos vistos no exemplo anterior, como o `alert()`, `createElement()`, `createTextNode()` e `appendChild()`, existem muitos métodos e propriedades que são utilizados para manipular o DOM e fazer a ligação entre os elementos HTML e os eventos desejados, que iniciam o *site* por meio de comportamentos.

Veja, no exemplo de código a seguir, como funciona a manipulação dos elementos a partir de uma instrução de carregamento de `script`, inserida no evento `onclick` da tag `input`, que captura o texto digitado pelo usuário no campo de texto e apresenta o elemento `p` quando o usuário clica no botão com o texto “Clique aqui!”.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="utf-8"/>
    <title>Título da Página na Aba</title>
    <script>
        var capturado = "";
        function capturar() {
            capturado = document.getElementById('valor').value;
            document.getElementById('valorDigitado').innerHTML =
capturado;
        }
    </script>
</head>
<body>
    <h1>Título da Página</h1>
```

```
<input type="text" id=valor>
<input type="submit" onclick="capturar()" value="Clique
aqui!">
<p id="valorDigitado"></p>
</body>
</html>
```

Propriedades fundamentais e tipos de dados

Por intermédio do objeto document também é possível acessar inúmeras propriedades. Veja, no Quadro 1, algumas das principais propriedades que podem ser acessadas por esse objeto.

Quadro 1. Propriedades que podem ser utilizadas através do objeto document

documentElement	Recupera o elemento raiz <html> do documento HTML.
getElementById	Retorna um elemento que usa o atributo ID correspondente.
createElement	Cria um novo nó de elemento para a página.
createAttribute	Cria um novo nó de atributo para a página.
createTextNode	Cria um novo nó de texto para a página.
getElementsByName	Recupera uma lista de elementos com o nome em questão.
appendChild	Insere um novo elemento filho.
removeChild	Remove o elemento filho em questão.
parentNode	Retorna o nó pai de um dado nó.

Fonte: Adaptado de Medeiros (2013).

Como pode-se perceber, grande parte do código que é utilizado com o DOM diz respeito à manipulação de páginas Web, ou seja, documentos HTML. É comum usar a referência dos elementos, como nós do DOM, já que no documento HTML, cada nó é um elemento. Veja, no Quadro 2, alguns desses tipos de dados.

Quadro 2. Tipos de dados

Document	Objeto raiz do documento.
Node	Todo objeto localizado em um documento HTML é um nó (em inglês, <i>node</i>), de algum tipo. Como visto no Quadro 1, o objeto pode ser um nó de elemento, um nó de atributo ou um nó de texto.
Element	O tipo do objeto Element é baseado no nó. Refere-se a um elemento ou a um nó do tipo elemento, retornado por um membro da API do DOM. Ao invés de dizer que o método <code>document.createElement()</code> retorna uma referência de um objeto para um nó, costuma-se dizer que o método retorna um elemento.
NodeList	O objeto NodeList é uma lista de elementos na qual seus itens podem ser acessados por meio de um índice, como no caso de <code>list.item(1)</code> ou <code>list[1]</code> . Ambos são equivalentes.
Attribute	Atributos (do inglês <i>attribute</i>) são nós no DOM, assim como os elementos, embora sejam menos utilizados do que eles.
NamedNodeMap	Este tipo de objeto é como a lista de nós; entretanto, os itens podem ser acessados tanto pelo nome quanto pelo seu índice, embora esse último caso seja uma conveniência para a enumeração, já que estes não ficam em uma ordem particular na lista. Este tipo de objeto tem um método <code>item()</code> para esse propósito, que pode ser usado, também, para adicionar e remover itens dessa lista.

Fonte: Adaptado de Franklyn (2019).

Além disso, é preciso ter em mente que existem terminologias para alguns desses tipos, como no caso de um nó de atributo, que pode ser chamado simplesmente de atributo, bem como na lista de nós DOM, que pode ser chamada apenas de lista de nós.

Veja, no tópico a seguir, como funcionam as interfaces do DOM.

Interfaces do DOM: HTML, CSS e Core

Segundo Franklyn (2019, documento *on-line*), muitos objetos DOM implementam, simultaneamente, diversas interfaces, como no caso do objeto que

representa o elemento `form`, que recebe a propriedade `name` da interface `HTMLFormElement`, e a propriedade `className`, da interface `HTMLElement`. Os objetos `document` e `window` são os objetos das quais as interfaces geralmente são as mais utilizadas na programação envolvendo o DOM. Em outras palavras, o objeto `window` representa algo como o navegador, enquanto o objeto `Document` é a própria raiz do documento da página Web.

Para Champion *et al.* (1998, documento *on-line*), as funcionalidades Core são suficientes para permitir que desenvolvedores de *software* e de *scripts* acessem e manipulem conteúdos HTML e XML de páginas Web de *sites* e sistemas. A API Core do DOM também possibilita popular um objeto `Document` usando apenas chamadas da API DOM.

De acordo com Queiroz *et al.* (2017, documento *on-line*), um documento que contém código em HTML é descrito a partir da interface `HTMLDocument`. A especificação da linguagem HTML também descreve a interface `Document`. Além disso, um objeto que implementa a interface `HTMLDocument` dá acesso a inúmeros recursos de navegadores, como a janela ou a aba, nos quais uma página é implementada a partir da interface `Window`. O estilo, normalmente em CSS, é associada a ela devido à interface `Style`, sendo possível obter a história do navegador, relativa ao contexto de navegação em questão, por conta da interface `History`.

Devido às implementações das interfaces apresentadas anteriormente é possível utilizar os métodos dos objetos para adicionar o comportamento ao *site*, como no exemplo a seguir.

```
<!DOCTYPE html>
<html>
<head></head>
<body>
    <h1 id="id1">Meu Cabeçalho 1</h1>
    <button type="button" onclick="document.getElementById('id1').style.color = 'red'">
        Clique em mim!
    </button>
</body>
</html>
```

Neste exemplo, a página apresenta um título com o texto “Meu Cabeçalho 1”, na cor preta, cor padrão do texto, e um botão com o texto “Clique em mim!”, que muda a cor do título para vermelha ao ser clicado.



Referências

CASTIGLIONI, M. Meu HTML é semântico e o seu? *CollabCode*, Medium, [S. l.], 17 maio 2018. Disponível em: <https://medium.com/collabcode/meu-html-%C3%A9-sem%C3%A2ntico-e-o-seu-4e97c81c0c49>. Acesso em: 14 out. 2019.

CHAMPION, M. et al. *Document object model (Core) Level 1*. Cambridge; Sophia-Antipolis; Minato; Beijing: World Wide Web Consortium, 1 Oct. 1998. Disponível em: <https://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html>. Acesso em: 14 out. 2019.

FRANKLYN, R. Introdução ao DOM. *MDN Web Docs*, Mountain View, 5 set. 2019. Disponível em: https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM/Introdu%C3%A7%C3%A3o#DOM_interfaces. Acesso em: 14 out. 2019.

INTRODUCTION to DOM - JavaScript Course #09. [S. l.: S. n.], 2019. 1 vídeo (28 min 4 s). Publicado pelo canal Curso em Vídeo. Disponível em: <https://www.youtube.com/watch?v=WWZX8RWLxIk>. Acesso em: 14 out. 2019.

JAVASCRIPT HTML DOM Document. *W3Schools*, Sandnes, 2019. Disponível em: https://www.w3schools.com/js/js_htmldom_document.asp. Acesso em: 14 out. 2019.

MALDONADO, L. Entendendo o DOM (Document Object Model): o DOM explicado de uma maneira fácil. *Tableless*, [S. l.], 8 fev. 2008. Disponível em: <https://tableless.com.br/entendendo-o-dom-document-object-model>. Acesso em: 14 out. 2019.

MEDEIROS, H. Trabalhando com DOM em JavaScript. *DevMedia*, Rio de Janeiro, 2013. Disponível em: <https://www.devmedia.com.br/trabalhando-com-dom-em-javascript/29039>. Acesso em: 14 out. 2019.

NASCIMENTO, F. Começando com o desenvolvimento Front-end. *Alura*, São Paulo, 21 mar. 2019. Disponível em: <https://www.alura.com.br/artigos/comecando-com-o-desenvolvimento-front-end>. Acesso em: 14 out. 2019.

QUEIROZ, W. C. et al. Modelo de Objeto de Documento (DOM). *MDN Web Docs*, Mountain View, 17 jun. 2019. Disponível em: https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM. Acesso em: 14 out. 2019.

**Encerra aqui o trecho do livro disponibilizado para
esta Unidade de Aprendizagem. Na Biblioteca Virtual
da Instituição, você encontra a obra na íntegra.**

Conteúdo:

