

# PROGRAMAÇÃO

## FRONT END II

Maikon Lucian Lenz



**s a  
g a h** SOLUÇÕES  
EDUCACIONAIS  
INTEGRADAS

# **Manipulando *browser object model*, *cookies* e temporizadores em JavaScript**

## **Objetivos de aprendizagem**

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

- Definir *browser object model*.
- Reconhecer os principais objetos do *browser object model*.
- Implementar *cookies* e temporizadores em JavaScript.

## **Introdução**

O *browser object model* não trata de um elemento padronizado, mas de livre implementação de cada um. Mesmo assim, há um grande denominador comum entre o modelo de cada navegador, e muitas funções estão presentes nos principais interpretadores disponíveis no mercado. Os objetos podem incluir desde temporizadores até lista de quadros da página, ou, ainda, o histórico do navegador. Por ser do topo da hierarquia de objetos em um navegador, o *browser object model* tem acesso a características próprias do navegador e da máquina que o utiliza, ao contrário do objeto `document`, vinculado diretamente à página em si.

Neste capítulo, você verá a definição de *browser object model*, reconhecerá seus principais objetos, além de estudar sobre como implementar *cookies* e temporizadores em JavaScript.

## ***Browser object model***

O núcleo de uma implementação em JavaScript para navegadores observa o padrão ECMAScript, mas existem outras partes não incluídas nele e que são

frequentes nesse tipo de sistema, como o *browser object model* (BOM) – ou modelo de objeto do navegador. Na verdade, o ECMAScript não restringe a implementação da linguagem que pode adicionar particularidades para além do núcleo padronizado (ECMA INTERNATIONAL, 2015).

Os navegadores modernos mais se parecem com um sistema operacional básico do que com os primeiros navegadores unicamente destinados à exibição de um documento (FLANAGAN, 2013).

Uma vez que o objeto é o tipo fundamental da linguagem JavaScript, é natural que os navegadores – que correspondem a uma parcela considerável dos ambientes hospedeiros – disponibilizem objetos, sejam controlados e se comuniquem por meio deles.

O BOM é um objeto que abrange os elementos não pertencentes à página em si, como: janela, histórico, barra de endereço, barra de *status* e tantos outros. Assim como qualquer outro objeto, o BOM é um conjunto de chaves e valores que representa as propriedades e os métodos do navegador. Já que o BOM não é padronizado pela linguagem, cada implementação pode desenvolver o seu próprio modelo de objeto. Devido a isso, deve-se ter bastante cuidado ao utilizar esse objeto, uma vez que o comportamento pode diferir de navegador para navegador. Na maioria dos navegadores, o BOM é acessível por meio de objeto denominado `window`, pelo qual grande parte das funcionalidades e propriedades é compartilhada, apesar de inexistir padronização.

O `window` pode ser referenciado pela palavra reservada `this`, em outras implementações, costuma referir-se ao objeto `global`, como era o caso do interpretador Node.js. Dessa forma, todos elementos globais, sejam objetos, variáveis ou funções, são filhos do objeto global `window` e podem ser acessados diretamente pelo seu nome quando o contexto for o mesmo do `global` ou por meio da notação ponto em conjunto com o objeto `window`, como no seguinte exemplo (FLANAGAN, 2013).



### Exemplo

```
window.obj = { prop1: 1 }
console.log(obj.prop1)           // escreve 1
console.log(window.obj.prop1)    // escreve 1
```

O objeto `window` assume todas as demais janelas abertas a partir dele como de mensagem – janelas de *pop-up*. Essas janelas adicionais estão associadas a `window`. Na verdade, são objetos membros dele e, como tal, fazem parte do BOM.

### Fique atento

O BOM e o objeto `window` não são sinônimos, mas, em geral, esse último costuma ser a principal implementação utilizada nos navegadores que utilizam esse conceito.

O objeto modelo do navegador é o ponto de partida de um sistema Web para a linguagem JavaScript. Qualquer declaração de variável, objeto ou função será um elemento novo e inferior nessa hierarquia.

Mesmo sem um padrão determinado, o BOM pode organizar os objetos filhos em seis categorias principais de objetos dependes, sem mencionar as propriedades e funções próprias desse.

## Objetos do BOM

Diretamente ligados ao objeto `window`, existem propriedades e métodos de manipulação da janela em si.

O tamanho e a posição da janela e seu documento podem ser controlados por meio de funções de:

- redimensionamento absoluto (`resizeTo`) ou relativo (`resizeBy`), para modificar largura e altura da janela, movendo o ponto da extremidade inferior direita;
- mudanças de posição absoluta (`moveTo`) ou relativa (`moveBy`), a partir do ponto referente à extremidade superior esquerda da janela;
- rolagem de página absoluta (`scrollTo`) ou relativa (`scrollBy`), para alterar a posição em que uma página maior que a tela é exibida.

Também é possível abrir ou fechar novas janelas utilizando:

- gerador de mensagens de alerta simples (`alert`), informando uma *string* a ser exibida em conjunto com um botão de finalização;
- mensagens de confirmação (`confirm`), contendo botão de confirmação, que retornará `true`, e de cancelamento, que retornará `false`;
- nova janela de diálogo para receber valores do usuário (`prompt`);
- abertura de novas janelas ou abas (`open`) a depender de como o interpretador implementou a função, normalmente informando características como página, dimensões, entre outros;
- fechamento da janela atual (`close`).

Outras funções comuns incluem, ainda, controle de foco, utilizando `focus` para direcioná-lo à janela, e `blur`, para tirá-lo da janela atual, além de interrupção de carregamento de página (`stop`).

Obviamente, esses métodos e funções controlam uma série de propriedades, dentre as quais as mais frequentes estão no Quadro 1, a seguir.

**Quadro 1.** Propriedades do objeto `window`

Propriedade	Descrição
<code>closed</code>	Propriedade booleana, igual a <code>true</code> quando a janela estiver fechada.
<code>console</code>	Acesso ao objeto console do navegador.
<code>defaultStatus</code>	Define ou retorna o texto-padrão da barra de <code>status</code> .
<code>document</code>	Retorna o objeto que armazena o documento da página.
<code>frameElement</code>	Retorna o quadro ( <code>&lt;iframe&gt;</code> ) no qual a página atual está inserida.
<code>frames</code>	Retorna todos os quadros ( <code>&lt;iframe&gt;</code> ) da janela.
<code>history</code>	Retorna o objeto que armazena o histórico.
<code>innerHeight</code>	Retorna a altura da área de conteúdo, incluindo as barras de rolagem.
<code>innerWidth</code>	Retorna a largura da área de conteúdo, incluindo as barras de rolagem.

(Continua)

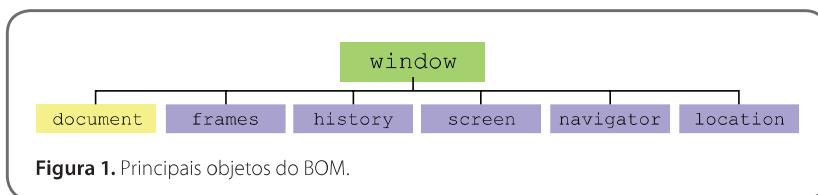
(Continuação)

**Quadro 1.** Propriedades do objeto window

Propriedade	Descrição
length	Retorna a quantidade de quadros (<iframe>) da janela.
localStorage	Utilizado para armazenar dados no formato chave-valor, de forma persistente.
location	Retorna a localização da janela.
name	Define ou retorna o nome da janela.
navigator	Retorna o objeto que contém o navegador.
opener	Retorna a referência à janela que criou a atual.
outerHeight	Retorna a altura total da janela.
outerWidth	Retorna a largura total da janela.
pageXOffset ou scrollX	Retorna a quantidade de pixels rolados da esquerda à direita da página atual.
pageYOffset ou scrollY	Retorna a quantidade de pixels rolados do topo para o baixo da página atual.
parent	Retorna a janela parente da atual.
screen	Retorna o objeto referente à tela da janela.
screenLeft ou screenX	Retorna a coordenada horizontal da janela em relação à tela.
screenTop ou screenY	Retorna a coordenada vertical da janela em relação à tela.
sessionStorage	Utilizado para armazenar dados no formato chave-valor, por uma sessão.
self	Retorna o objeto da janela atual.
status	Define ou retorna o texto da barra de status da janela.
top	Retorna a primeira janela na hierarquia.

*Fonte:* Adaptado de The Window... (2019).

O objeto `window` é apenas o topo de uma hierarquia de outros objetos comuns nos navegadores. Deve-se frisar sempre que não há um padrão para essa implementação, mas existem muitos objetos em comum no BOM entre os navegadores. Os principais e mais comuns, apresentados no Quadro 1, podem ser vistos isoladamente na Figura 1, a seguir.



**Figura 1.** Principais objetos do BOM.

Cada um desses objetos apresentados expõe uma parte do navegador ou da página atual e apresenta suas próprias funções e propriedades.

## Objeto `document`

O objeto de maior importância no desenvolvimento Web, contém a página HTML carregada pela janela do navegador, sendo ela o foco principal não apenas do programador, mas do usuário.

Assim como o BOM, o objeto `document` também é costumeiramente referenciado pelo seu acrônimo DOM (do inglês *document object model*). É por meio do DOM que se pode acessar e modificar o conteúdo da página que está sendo exibida pelo navegador (FLANAGAN, 2013).

Como qualquer outro objeto, o DOM apresenta métodos e propriedades próprios. A sua estrutura organiza os elementos de uma página por categorias, facilitando encontrar imagens ou formulários presentes, por exemplo.

## Objeto `frames`

Este objeto lista todos os quadros existentes no objeto `window` na forma de um *array* de objetos. Assim, cada um dos quadros é acessado por meio de um índice, ordenado na ordem em que aparecem cada um dos quadros, conforme exemplo a seguir.



## Exemplo

```
const quadros = window.frames
for (let i in quadros)
{
  console.log(quadros[i])    // exibe o objeto de cada quadro
  no console do navegador
}
```

Os quadros correspondem às *tags* <frame> e <iframe> do padrão HTML. Porém, com o novo padrão HTML5, os elementos <frame> deixaram de existir, mantendo apenas os <iframe>.

## Objeto history

Este mantém o histórico de páginas acessadas pela janela, sendo possível:

- retornar às páginas anteriores por meio da função `back()`;
- avançar para páginas mais recentes no histórico, com a função `forward()`;
- carregar novas páginas a partir de um endereço *uniform resource locator* (URL, que em português significa “localizador uniforme de recursos”) — com a função `go()`;
- retornar a quantidade de páginas já acessadas por meio da propriedade `length`.

## Objeto screen

Este armazena informações referentes à tela do usuário que visualiza a página como: quantidade de pixels em altura (`availHeight`) e largura (`availWidth`) disponíveis, excluindo a barra de tarefas do sistema operacional; resolução de cores utilizada (`pixelDepth` e `colorDepth`); altura (`height`) e largura (`width`) totais da tela.

## Objeto navigator

Este contém informações e métodos referentes ao navegador, dados sobre o aplicativo: nome (`appName`), versão (`appVersion`), plataforma (`platform`); e nome do motor utilizado pelo aplicativo (`product`). Além disso, disponibiliza algumas configurações: permissão para armazenar *cookies* (`cookieEnabled`); localização geográfica do usuário (`geoLocation`); idioma utilizado (`language`); e se o navegador está *on-line* (`onLine`). Por fim, a maioria dos navegadores contém um cabeçalho de informações sobre o usuário do navegador, que é enviado para o servidor (`userAgent`).

## Objeto location

O último objeto apresentado é o `location`, no qual constam informações quanto ao nome do servidor, ao protocolo, à porta e ao endereço da página que estão sendo acessados na janela, retornados pelas propriedades: `origin`, com protocolo, nome e porta; `host`, retorna nome e porta; `hostname`, apenas o nome; `protocol`, apenas o protocolo; `port`, apenas a porta; `pathname`, apenas o caminho que precede o arquivo do endereço; `href`, o endereço completo; `hash`, retorna os valores ancorados após o endereço, simbolizados pelo prefixo “#”; e `search`, que retorna os parâmetros de consulta inseridos após o endereço, simbolizados pelo prefixo “?”.

O exemplo a seguir apresenta o uso da propriedade `search`. Exemplo similar poderia utilizar propriedade `hash` também, mas, nesse caso, o caractere de referência seria o “#”.

Na próxima seção, serão estudados os temporizadores e *cookies*. Diretamente vinculado ao objeto `window`, existem funções de uso frequente, como os dois tipos de temporizadores disponíveis: o periódico e de intervalos; e o de atraso ou de evento único.

Os *cookies*, objeto pertencente ao `document`, têm objetivo de armazenar, temporariamente, informações relevantes para o *site*, usuário ou servidor, que permitam uma melhor interação do aplicativo com o sistema e o usuário em si.



## Exemplo

```
// http://www.sagah.com.br/index.html?valor=20
const resultado = window.location.search
let valor = 0;

// separa a string em duas no caractere '='
const splitted = resultado.split('=')

// verifica se o texto anterior ao '=' é '?valor'
if (splitted[0] === '?valor')
{
    // converte a parte após o símbolo de '=' para inteiro
    valor = Number.parseInt(splitted[1])
}

// exibe 25, o resultado de 20 + 5
console.log(valor + 5)
```

## Cookies e temporizadores em JavaScript

Entre os objetos disponibilizados pelo BOM, duas funções se destacam: a memorização de dados por meio de *cookies* e a temporização.

Os *cookies* são pequenos arquivos de texto utilizados para armazenar dados referentes à navegação do usuário em uma determinada página, para que, quando ele acesse novamente a página, algumas das informações já estejam disponíveis, possibilitando interações mais rápidas e carregamento mais eficiente. Para isso, utilizam-se de dados já armazenados no passado e que se sabe, mediante interação com o servidor, que não foram alterados.

Esse recurso pode, inclusive, armazenar informações de *login*, como nome de usuário e senha, para que não seja necessário fornecer os dados toda vez que a página for acessada.

Cada *cookie* é referenciado por um nome e está associado diretamente a um endereço de uma página específica, ficando armazenado no lado do cliente. Porém, podem ser acessados e manipulados pelo lado do servidor por meio do envio de dados do *cookie* entre as partes (FLANAGAN, 2013).

Por se tratar de uma API antiga, a manipulação de *cookies* diverge bastante da usual. Qualquer procedimento envolve a leitura ou escrita da propriedade `window.document.cookie`.



### Exemplo

```
// cookies permitidos?  
if (window.navigator.cookieEnabled === true)  
{  
    // cria um novo cookie de nome login  
    window.document.cookie = 'login=alberto'  
    // cria um novo cookie de nome pass  
    window.document.cookie = 'pass=14bis'  
}
```

No exemplo anterior, caso o navegador esteja configurado para permitir a criação de *cookies* (`cookieEnabled`), dois novos *cookies* são criados – isso, claro, se ainda não existirem *cookies* com o mesmo nome vinculados ao endereço daquela página. Nesse último caso, os *cookies* existentes seriam substituídos pelos novos.

O recurso permite, ainda, a especificação de alguns atributos, como o tempo de vida do *cookie*, já que, por padrão, *cookies* duram apenas enquanto a mesma sessão do navegador existir. Para contornar essa restrição, o atributo `expires` determina a data que, uma vez ultrapassada, excluirá automaticamente o *cookie*, como o caso no exemplo a seguir.



## Exemplo

```
// cookies permitidos?  
if (window.navigator.cookieEnabled === true)  
{  
    // data/hora atuais  
    Let hoje = new Date()  
  
    // data/hora mais 1 hora  
    Let expira = new Date(hoje.getTime() + 3600000)  
  
    // concatena o cookie e seu atributo "expire" com o  
    valor de data/hora desejada  
    window.document.cookie = 'idioma=português; expires=' +  
        expira.toUTCString()  
}
```

Nesse exemplo, o *cookie* de nome “idioma” seria criado com o valor português e um prazo de validade de uma hora. Uma vez decorrido esse prazo, o valor seria eliminado.

Outros atributos podem incluir a especificação de um caminho que deve estar presente na URL, para que o *cookie* seja enviado por meio de path, e a determinação de um prazo em segundos, max-age.

Já os temporizadores dentro do BOM podem ser divididos em duas funções: setTimeout e setInterval.

O método setTimeout recebe dois parâmetros: o primeiro do tipo função, para referenciar a função que deverá ser executada ao cumprir o tempo especificado; e o segundo será o próprio tempo em milissegundos que o sistema deverá aguardar até chamar a função.

Para o método setInterval, os parâmetros são os mesmos e apresentam as mesmas finalidades da função setTimeout. O único diferencial é que, nesse caso, não estão sendo determinado um atraso para a execução da função uma única vez, e sim um intervalo de tempo em que ela deverá ser chamada periodicamente.

No exemplo a seguir, a função `mostrarTeste` é invocada após cinco segundos, mostrando a mensagem “Teste” no console do interpretador.



### Exemplo

```
Function mostrarTeste()
{
    console.log('Teste')
}
window.setTimeout(mostrarTeste, 5000)
```

É possível interromper ambos os temporizadores, informando a instância criada pelo acionamento do temporizador às respectivas funções: `clearTimeout` e `clearInterval`. O seguinte exemplo armazena a referência ao temporizador criado pela função `setInterval`, para interrompê-lo após cinco segundos contados por outro temporizador, utilizando o `setTimeout`. Logo, a função `mostrarTeste` será executada repetidamente a cada 1000 ms até que o `clearInterval` seja invocado pelo `timeout`.



### Exemplo

```
Function mostrarTeste()
{
    console.log('Teste')
}

Let timer1 = setInterval(mostrarTeste, 1000)
Let timer2 = setTimeout(function teste() {
    clearInterval(timer1);
    console.log('Fim') }, 5100)
```

Como você pôde ver, o modelo BOM é bastante amplo e, apesar de muito utilizado, não possui um padrão definido. No entanto, os principais navegadores mantêm funções e objetos similares entre si, o que facilita a integração e o desenvolvimento de aplicativos universais.

Devido ao tamanho e à complexidade do objeto atualmente, o BOM costuma ser subdividido, categorizando e vinculando cada objeto a uma parte do navegador, da tela ou da janela. Uma vez que o objeto de maior interesse de um aplicativo de navegação Web é a própria página que está sendo acessada, é natural que um determinado objeto que o englobe tenha algum destaque. É o caso do objeto `document`, que concentra todas as informações referentes à página em si.



## Referências

ECMA INTERNATIONAL. *ECMAScript® 2015 Language Specification*. 6. ed. Geneva: Ecma, 2015. 545 p. Disponível em: <https://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf>. Acesso em: 22 out. 2019.

FLANAGAN, D. *JavaScript: o guia definitivo*. 6. ed. Porto Alegre: Bookman, 2013. 1080 p.

THE WINDOW Object. *W3Schools*, Sandnes, 2019. Disponível em: [https://www.w3schools.com/jsref/obj\\_window.asp](https://www.w3schools.com/jsref/obj_window.asp). Acesso em: 22 out. 2019.

## Leituras recomendadas

JAVASCRIPT. *MDN Web Docs*, Mountain View, 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 22 out. 2019.

SANDERS, B. *Smashing HTML5: técnicas para a nova geração da web*. Porto Alegre: Bookman, 212. 368 p.

SIMPSON, K. *JavaScript and HTML5 now: a new paradigm for the open web*. Sebastopol: O'Reilly, 2012. 12 p.

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

Conteúdo:

