

MATERIAL EXTRA

Window, Location e
History



WINDOW

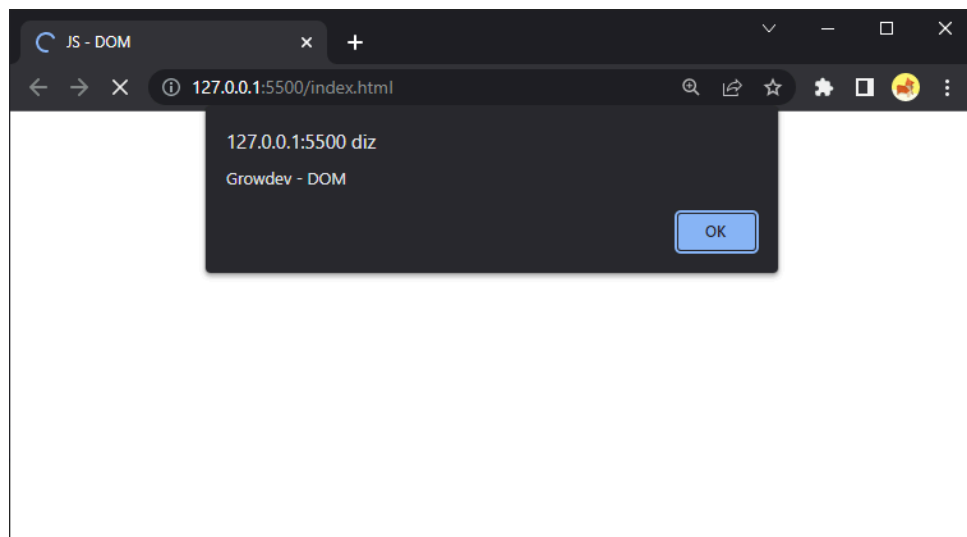
O item `window` do DOM permite manipular a página web a partir da janela de visualização do navegador. Seu acesso é feito a partir do objeto `window` do Javascript. Por ser a raiz do DOM, o objeto `window` possui todos os outros recursos do DOM anexados a si. Porém, alguns recursos possuem atalhos no Javascript para facilitar a escrita do código.

Neste sentido, podemos dividir o DOM em duas partes: a parte da **janela de visualização** a partir do objeto `window`; e a parte do **documento HTML** a partir do objeto `document`. Com os métodos específicos de `window`, é possível mudar a URL da página, gerar mensagens de alerta e escrever dados no console do navegador.

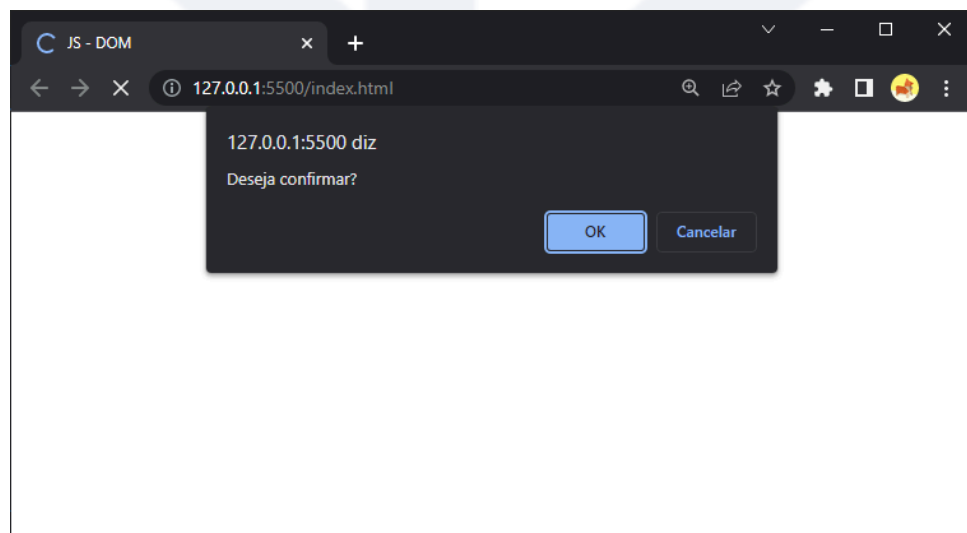
Alert

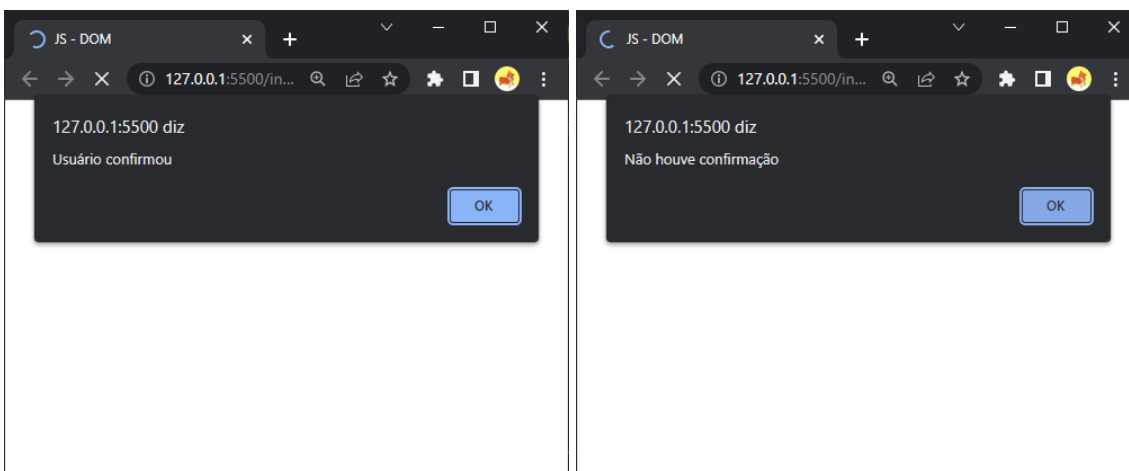
O comando `alert` faz com que o navegador chame um pop-up na página com uma mensagem definida em código. A função é chamada através do comando `window.alert` ou pelo atalho `alert`. O parâmetro de `alert` é a mensagem (em string) a ser exibida na tela.

```
// Chama um alert na página:  
window.alert("Growdev - DOM")  
  
// Ou usando o atalho:  
alert("Growdev - DOM")
```



É importante notar que o comando `alert` bloqueia a execução do código Javascript até que o usuário clique no botão "ok". Há uma variação do `alert` que pede uma confirmação: o `window.confirm`. Neste caso, abre-se um alerta com dois botões ("ok" e "cancelar") de modo que a função retorna `true` ou `false` para o clique em ok ou cancelar respectivamente.





Console

Quando usamos os comandos, a execução do Javascript é bloqueada até que uma ação seja tomada. Isso dificulta o uso dessas mensagens para fins de debug. Para este motivo, o window possui um conjunto de funções dentro do objeto console, em que é possível escrever mensagens no **console dos navegadores**.

Além disso, com `alert` e `confirm`, só podemos informar ao usuário textos ou números. Estruturas de dados como objetos e listas do Javascript não são exibidos corretamente. No console, porém, podemos exibir esses dados com um nível de detalhamento maior, possibilitando um **debug** eficiente.

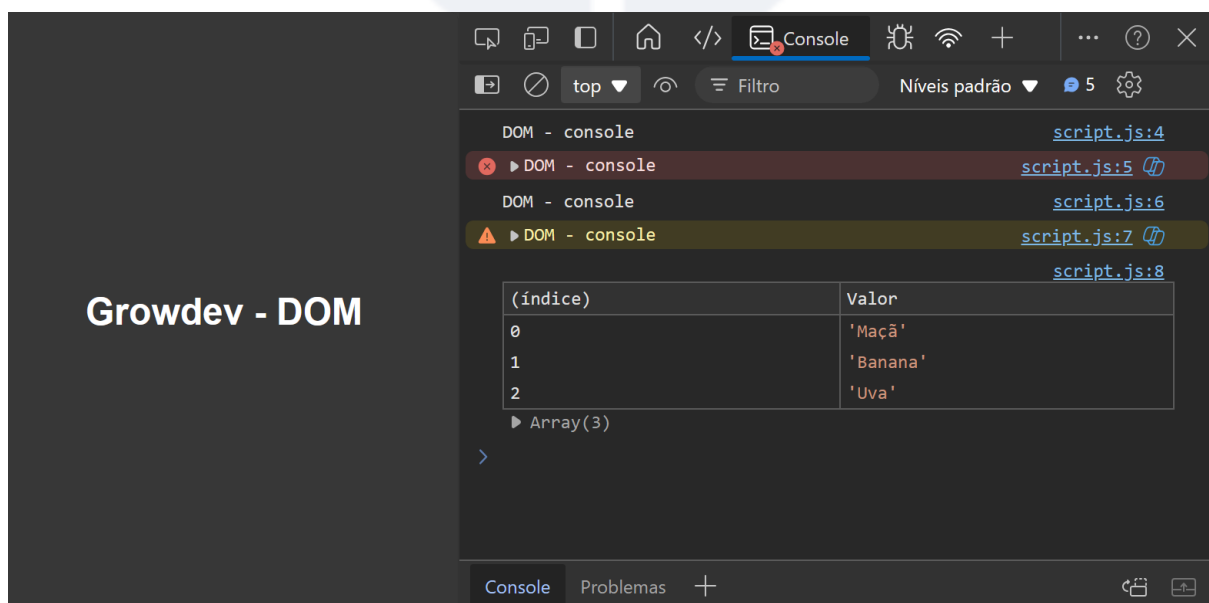
Dentro do escopo do console, existem diferentes métodos para imprimir mensagens ou dados. Os quatro principais são:

- **log**: o mais usado, apenas imprime a informação na tela;
- **info**: similar ao log, porém possui um valor semântico de informação;

- **error:** indica uma mensagem de erro e aparece no console com a cor vermelha;
- **warn:** indica um alerta de código e aparece no console com a cor amarela.
- **table:** permite visualizar dados em uma tabela organizada no console do navegador. Ele é especialmente útil quando você está lidando com arrays de objetos ou matrizes, já que exibe as informações de uma maneira mais legível do que `console.log()`

```
const texto = "DOM - console"
const frutas = ["Maçã", "Banana", "Uva"]

console.log(texto)
console.error(texto)
console.info(texto)
console.warn(texto)
console.table(frutas)
```

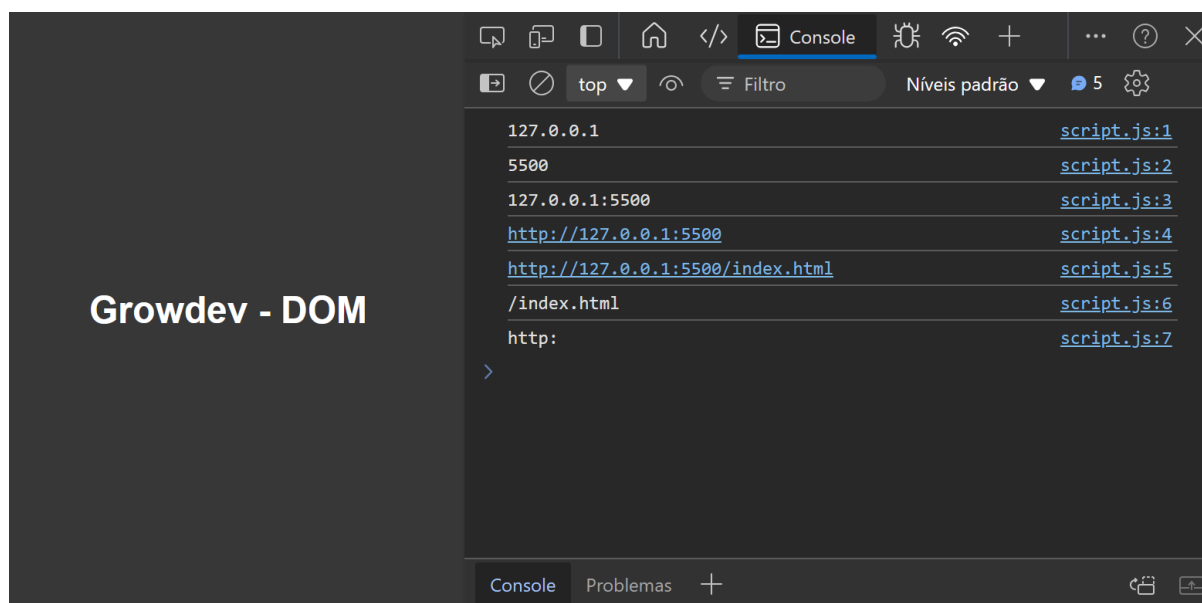


LOCATION

O objeto `location` está localizado como filho direto de `window` e pode ser acessado pelo comando `window.location`. O atalho `location` também pode ser usado. Seu uso é útil para identificar atributos quanto à **URL** e aos dados da **página** acessada. Os principais atributos exibem as seguintes informações:

- **hostname**: o domínio da URL acessada;
- **port**: a porta da URL acessada;
- **host**: o domínio e a porta da URL acessada;
- **origin**: a URL de origem da URL acessada (sem o caminho da página);
- **href**: a URL completa, contendo a origem e o caminho da página;
- **pathname**: apenas o caminho da página a partir da origem;
- **protocol**: o protocolo de comunicação usado.

```
console.log(location.hostname)
console.log(location.port)
console.log(location.host)
console.log(location.origin)
console.log(location.href)
console.log(location.pathname)
console.log(location.protocol)
```



O `window.location` também apresenta métodos úteis para a manipulação de **navegação** da página. Os principais métodos são:

- **assign**: redireciona a página para a URL informada como parâmetro. Também pode ser usado um caminho como parâmetro, ao invés de uma URL completa. Neste caso, a origem se mantém a mesma e a navegação ocorre dentro do mesmo site.
- **replace**: redireciona a página para a URL informada como parâmetro mas remove a página atual do histórico de navegação.
- **reload**: recarrega a página atual.

Além da função `assign`, o redirecionamento é usualmente feito através da propriedade `location.href`. Basta atribuir um novo valor para a propriedade que o DOM deve se encarregar de fazer o redirecionamento do mesmo jeito que o `assign`.

```
// Redireciona para a URL:
location.assign("http://localhost:8081/pagina2.html");
// Ou usando um path como parâmetro:
location.assign("/pagina2.html");
// Ou usando o href:
location.href = "/pagina2.html";

// Redireciona para o path removendo a página
// atual do histórico de navegação:
location.reload("/pagina2.html");

// Atualiza a página:
location.reload();
```

HISTORY

O objeto `history` é filho de `window` e serve como um manager (gerenciador) do **histórico do navegador**. Seu acesso pode ser feito via comando `window.history` ou pelo atalho `history`. Com esse recurso, é possível navegar através do histórico bem como manipulá-lo.

As principais funções disponíveis em `history` são:

- **back**: retorna uma página no histórico (mesmo funcionamento do botão “voltar” do navegador);
- **forward**: avança uma página no histórico (mesmo funcionamento do botão “avançar” do navegador);
- **go**: retorna ou avança N páginas no histórico baseado em um número passado como parâmetro. Se o número é positivo então avança; se não volta.

```
// volta no histórico
history.back();
// avança no histórico
history.forward();
// avança duas páginas no histórico
history.go(2);
// volta três páginas no histórico
history.go(-3);
```

REFERÊNCIAS

1. [Introdução ao DOM - APIs da Web | MDN](#)
2. [Window - Web APIs | MDN](#)
3. [Document - Web APIs | MDN](#)

BONS ESTUDOS