Coisas sobre o console do seu navegador que vão mudar a sua vida

um e-book ***produzido por:***

CodePrestige

Agradecimentos

Seja muito bem-vindo!

Os navegadores atuais possuem uma série de funcionalidades que facilitam as nossas vidas como desenvolvedores. Entretanto, poucos conhecem o seu verdadeiro poder. Neste e-book, te mostraremos alguns dos truques mais fantásticos e pouco explorados nos navegadores da atualidade. Tenho certeza de que você irá se impressionar!

Agora sem mais delongas, boa leitura e bons códigos!

Diego Martins de Pinho
Cofundador da Code Prestige

Sumário

Vamos falar sobre o console

O console do navegador é uma ótima ferramenta que nos ajuda diariamente com:

- Debug do código (o famoso print “passei por aqui”)
- Executar js *on the fly*
- Observar mensagens de erro/warnings do seu programa e/ou navegador
- Etc

Mas o console vai muito além disso... Vamos dar uma olhada no que podemos fazer com ele.

Exibir mensagens

Geralmente utilizamos a função ***`log()`*** para exibir mensagens no console. Mas na verdade há outras maneiras que podem ser mais adequadas. Temos outras três funções para exibir mensagens no console:

1. ***`info()`***
2. ***`warn()`***
3. ***`error()`***

Elas funcionam de forma semelhante, a diferença está na forma como são exibidas no navegador.



Exibir mensagens

Para cada tipo de mensagem, temos uma visualização diferente:

Exemplo:

```
> console.log('mensagem de boas');
  console.info('mensagem de info');
  console.warn('mensagem de warn');
  console.error('mensagem de erro');
  mensagem de boas
  mensagem de info
▲ ▶ mensagem de warn
⊗ ▶ mensagem de erro
  undefined
>
```

Atenção!

Para cada navegador, o texto é formatado de maneira diferente. Neste e-book, usaremos a visualização gerada pelo Google Chrome.

Arrays

Leve em conta este array de filmes:

```
var filmes = [
  {titulo: 'Mulher Maravilha', produtora: 'Warner Bros.'},
  {titulo: 'Homem Aranha', produtora: 'Marvel'},
  {titulo: 'Esquadrão Suicida', produtora: 'Warner Bros.'},
  {titulo: 'Deadpool', produtora: '20th Century Fox'},
];
```

Como poderíamos fazer pra ver os valores desse array no console? `console.log(filmes)` funcionaria, certo? Mas será que não tem um jeito melhor?

Exibindo arrays no console

Para exibir os itens de uma lista com vários objetos, podemos fazer uso da função ***table()*** do console. O navegador automaticamente irá montar uma tabela onde as propriedades são os valores das colunas:

Exemplo:

```
console.table(filmes);
```

(index)	titulo	produtora
0	"Mulher Maravilha"	"Warner Bros"
1	"Homem Aranha"	"Marvel"
2	"Esquadrão Suicida"	"Warner Bros"
3	"Deadpool"	"20th Century Fox"

▶ Array(4)

Agrupamento de mensagens

Dependendo do número de mensagens que estamos exibindo durante a execução dos nossos programas, o console pode ficar uma bagunça.

Às vezes queremos agrupar as nossas mensagens para que seja mais fácil encontrá-las quando foram exibidas no console.

Podemos fazer isso através da função ***`group()`***.



Agrupamento de mensagens

Exemplo:

```
> console.group('grupo 1');
  console.log('mensagem do grupo 1');
  console.group('grupo 2');
  console.log('mensagem do grupo 2');
  console.groupEnd('grupo 2');
  console.groupEnd('grupo 1');
▼ grupo 1
    mensagem do grupo 1
  ▼ grupo 2
      mensagem do grupo 2
< undefined
```

Também podemos usar a função ***groupCollapsed()***. Neste caso, o próprio nome diz, o grupo vem fechado.

Métricas

Quando queremos medir o tempo que o nossa função leva para ser executada, o que normalmente fazemos? Normalmente algo parecido com isso...

Exemplo:

```
> var inicio = performance.now();

  for (i = 0; i < 50000; ++i) {
      // do something
  }

  var final = performance.now();
  var tempo = final - inicio;
  console.log('tempo em ms', tempo);
  tempo em ms 1.7199999999720603
< undefined
>
```



Métricas

Uma alternativa interessante é usar o método ***`time()`*** do console.

Exemplo:

```
> console.time('loop');

  for (i = 0; i < 50000; ++i) {
    // não faz nada
  }

  console.timeEnd('loop');

  loop: 1.919921875ms
<- undefined
>
```

Construção de Strings

Por vezes temos a necessidade de unir várias informações em uma única String, como por exemplo:

Exemplo:

```
> const primeiroNome = 'Diego';
  const segundoNome = 'Pinho';
  const nomeCompleto = `${primeiroNome} ${segundoNome}`;
  console.log(nomeCompleto); // Diego Pinho
  Diego Pinho
< undefined
>
```

Isso funciona ok, mas podemos otimizar um dos passos...



Construção de Strings

Podemos usar templates literais diretamente na função ***log*** (e seus derivados):

Exemplo:

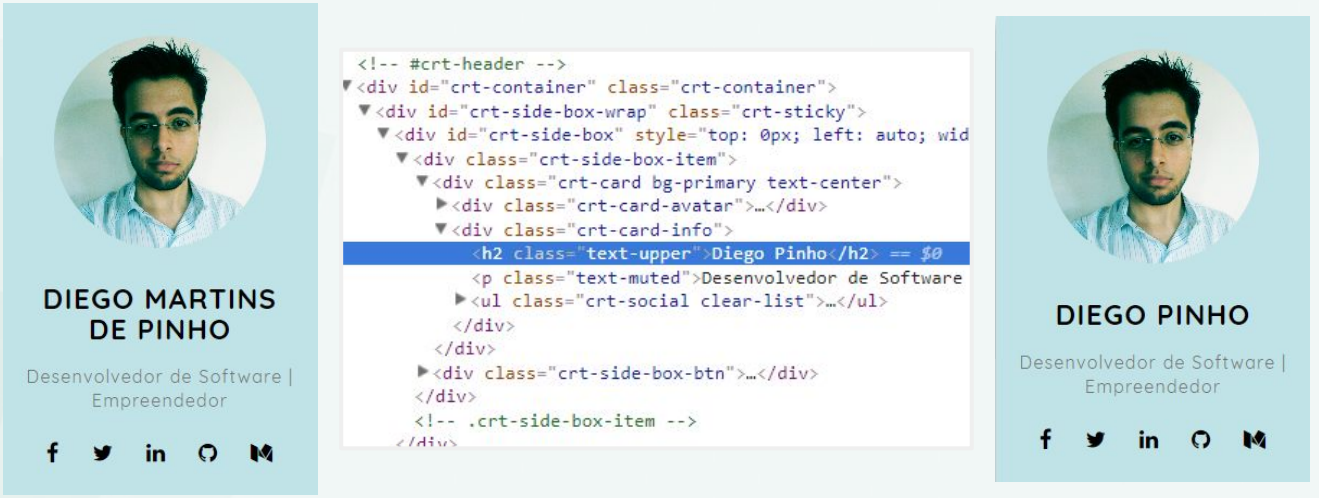
```
> const primeiroNome = 'Diego';
  const segundoNome = 'Pinho';

  console.log(`${primeiroNome} ${segundoNome}`);
  Diego Pinho
< undefined
>
```

Edição de páginas

O console é ótimo para conseguirmos alterar o conteúdo de um site *on the fly*.

Exemplo:





Edição de páginas

É possível alterar o conteúdo diretamente no site sem a inspeção de elementos utilizando o comando:

```
document.body.contentEditable=true
```

Com isso, já é possível alterar o conteúdo diretamente na página, sem a necessidade do console.



Testes

Existem mais alguns métodos interessantes. Um deles é o **Assert**. Podemos utilizá-lo para fazer alguma validação no código:

Exemplo:

```
> let verdadeiro = true;
  console.assert(verdadeiro,
  'Não vou aparecer');
  verdadeiro = false;
  console.assert(verdadeiro, 'Vou aparecer');
✖ ▶Assertion failed: Vou aparecer
  undefined
>
```



Limpeza

Depois de muito tempo utilizando o console, ele começa a ficar com informações demais.

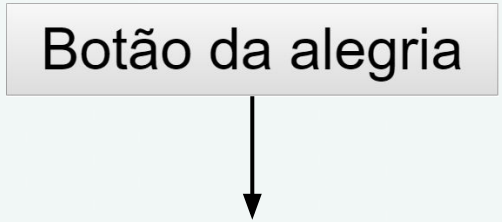
Para limpá-lo, há 3 opções:

1. Botão “limpar”
2. método ***clear()***
3. Ctrl + L

Monitoramento de eventos

Também podemos utilizar o console para monitorar eventos. Como por exemplo, o clique em um botão.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Teste</title>
</head>
<body>
  <button id="botao"
    type="button">Botão da alegria</button>
</body>
</html>
```




```
> monitorEvents($('#botao'), 'click');
< undefined
  click ▶ MouseEvent {isTrusted: true, screenX: 64, screenY: 112, clientX: 64, clientY: 22, …}
```

Monitoramento de eventos

Há várias maneiras de monitorar:

1. ***monitorEvents***($(‘selector’))
2. ***monitorEvents***($(‘selector’),’eventName’)
3. ***monitorEvents***($(‘selector’),[‘eventName1’, ….])

Para qualquer um deles, podemos desabilitar o monitoramento com o método ***unmonitorEvents***($(‘selector’))

Capturar eventos

Exemplo:

```
> getEventListeners($('#botao'));
< ▼ {click: Array(1)}
    ▼ click: Array(1)
      ▼ 0:
        ▶ listener: ƒ ()
          once: false
          passive: fal
          type: "click"
          useCapture: false
        ▶ __proto__: Object
          length: 1
        ▶ __proto__: Array(0)
      ▶ __proto__: Object
>
```

function () { console.log('fui clicado!'); }

Se não soubermos de antemão os eventos associados à um elemento, podemos usar o método **getEventListeners()** para descobrir!

Referências

Esta apresentação é baseada na série de artigos "*Você conhece o verdadeiro poder do console do seu navegador?*" publicado na nossa publicação do Medium.

1. https://medium.com/code-prestige/console-navegador-3f2434124eaf
2. https://medium.com/code-prestige/voc%C3%AA-conhece-o-verdadeiro-poder-do-console-do-seu-navegador-parte-2-efaa540c148

CodePrestige

Ensino de programação à distância

/CodePrestige

Confira outros e-books, vídeos e cursos nas nossas redes sociais!