

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Mini-curso de Introdução à Linguagem JavaScript



```
.ui-helper-hidden-accessible  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Prof. Jivago Medeiros <jivago@ic.ufmt.br>

O que é JavaScript?

- JavaScript (JS) é uma linguagem de programação interpretada, baseada em *script* e comumente utilizada em navegadores web (*client-side*).
 - É uma linguagem **multi-paradigma**, adotando diferentes paradigmas, entre esses, alguns aspectos de orientação a objetos. É fracamente tipada com tipagem dinâmica.
 - Proposta inicialmente para o **navegador** Netscape em 1995 e adotada parcialmente pelo Internet Explorer em 1996.

História

- 1994 – Lançamento do (Mosaic) Netscape
 - Era necessária uma linguagem que auxiliasse desenvolvedores
- Nome original: Mocha
 - Criada em 10 dias (!) por Brendan Eich, em Maio/95
 - Escolhido por Marc Andreessen, fundador da NetScape
- Setembro/95
 - Livescript
- Dezembro/95
 - JavaScript (jogada de marketing!)
 - Lançada junto com o Netscape 2.0

História

- 1996
 - Jscript (Microsoft) – versão do Javascript para IE
 - Problemas de compatibilidade
- 96-97
 - ECMA (European Computer Manufacturers Association)
 - Padrão ECMAScript
- 98 – ECMA2 , 99 – ECMA3, ECMA4 – Projeto Abandonado, 2009 – **ECMA5**, 2011 – **ECMA5.1**, 2015 – ECMA6, 2016 – ECMA7 (ECMAScript 2016), 2017 – ECMA8, 2018 – ECMA9...

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Mais história...

- No início, caiu em descrédito com os desenvolvedores por alguns anos principalmente pela incompatibilidade entre navegadores.
 - Alguns desenvolvedores alegavam também que era uma linguagem “para leigos”, dando origem a *scripts* “confusos/bagunçados” e pouco otimizados.
- Sua popularidade foi restaurada com o suporte a requisições assíncronas (AJAX) e surgimento de bibliotecas JavaScript, a citar: prototype e jQuery

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Inserindo no Navegador

- Os scripts JavaScript são inseridos nos documentos HTML utilizando a tag `<script>`
- Um script JS pode ser definido diretamente entre a tag, ex:

```
<script> alert('Olá'); </script>
```

- Ou ser oriundo de um arquivo externo, usualmente com a extensão .js, ex:

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;  
}  
<script src="meu-script.js"></script>
```

Variáveis

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

- Variáveis
 - Não precisamos dizer o tipo, exemplo:
 - **var** a = 123;
 - E a tipagem é dinâmica (podemos alterar o tipo da variável em tempo de execução):
 - a = “agora sou uma string”;

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Funções

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

- Funções

- Usa-se a palavra reservada **function** e não é necessário “tipar” o retorno e os parâmetros, ex:

```
function soma(a,b) {  
    var c = a+b;  
    return c;  
}
```

- Podemos definir uma função *"dentro de uma variável"*, e depois instanciá-la:

```
var A = function () {  
    console.log("Sou uma classe?");
```

```
b = new A();  
c = new A();
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```


Funções

- Na especificação ECMAScript 2015 foi introduzida funções lambdas, também conhecida em JavaScript com *arrow function*, exemplo:

```
var soma = (a,b) => a + b;
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0 0 0 0;  
position: absolute;  
width: 1px;
```

Prototype

- Como mencionado, JS é uma linguagem multi-paradigma cuja a orientação a objetos implementada é baseada em prototipação. Isso significa que podemos fazer alterações nas "classes" durante a execução e todas as instâncias recebem essas alterações, exemplo:

```
A.prototype.ola = function () {  
    alert("Olá Mundo!");  
}
```

```
c.ola();
```

Importante: classes, como conhecemos em outras linguagens como JAVA, foram introduzidas em JavaScript no ECMAScript 2015:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes>

Arrays

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

- Usualmente, declaramos arrays em JS de duas formas:

```
var arr = ["a", "b", "c", "d"];
```

ou

```
var arr = new Array("a", "b", "c", "d");
```

- Os valores do array podem ser acessados utilizando: `arr[0]`, `arr[1]`, etc **ou** ainda `arr.1`, `arr.2`, etc
- Sendo "uma variável um array", podemos utilizar métodos como:

```
push(), pop(), shift(), splice(), etc
```

- E propriedades como `.length`

Outros métodos e propriedades:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

Objetos

- Objetos podem possuir métodos e propriedades
 - A forma recomendada de declarar objetos é:
 - `var obj = {nome : “João”, idade : “21”, curso : “SI”};`
 - Os valores de um objeto pode ser acessados utilizando:
 - `obj.nome, obj.idade, obj.curso`
 - ou
 - `obj['nome'], obj['idade'], obj['curso']`
 - Objetos aceitam que as “propriedades sejam funções” (métodos)

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Objetos

```
var aluno = {  
    nome : "João",  
    idade : "21",  
    curso : "SI",  
    correr : function () {  
        if (this.idade <= 30) {  
            console.log(obj.nome+" Corre muito!");  
        }  
        else {  
            console.log(obj.nome+" Corre pouco!");  
        }  
    }  
}
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Manipulação do DOM

- O que é o DOM ?
 - *Document Object Model*
 - É a árvore dos elementos (objetos) renderizados (exibidos) pelo navegador.
 - Após o carregamento de um documento HTML o navegador gera o objeto *document* que contem todos os elementos da página.
- A linguagem JavaScript fornece um conjunto de métodos para a manipulação do DOM.

document.getElementById()

- O método `getElementById()` é utilizado para “pegarmos” (referenciarmos) um elemento no document (DOM) pelo seu atributo id, `<p id='paragrafo'>...</p>`, ex:
 - *document.getElementById('paragrafo');*
 - Era um dos principais métodos para manipulação do DOM antes do surgimento e popularização das bibliotecas JavaScript.

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

document.getElementById()

- Após “pegarmos” um elemento, podemos fazer “qualquer coisa” com ele, por exemplo:
 - Alterar estilo
 - Acessar / atualizar atributos
 - Remover do DOM
 - etc

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```


document.getElementById()

- Exemplo:

```
var elem = document.getElementById('paragrafo');
```



A partir desse instante, a variável **elem** faz referência ao elemento com id “paragrafo” que se encontra no DOM do documento HTML.

Assim, utilizando o **elem** é possível, por exemplo, retornar, ou alterar o texto do elemento ou mesmo retornar ou alterar a borda do elemento.



```
elem.textContent;  
elem.style.backgroundColor="#006600";
```

Importante!

- Sobre recuperar o valor de estilos utilizando JavaScript:
 - Em muitos casos (ex: não ter nenhuma propriedade atribuída) não é possível retornar o valor da propriedade CSS (estilo) utilizando *elem.style.propriedade*
 - Nesses casos devemos utilizar *window.getComputedStyle()*

```
var par = document.getElementById("paragrafo");  
  
//primeiro pegamos o conjunto de 'estilos computados'  
//para um elemento, neste caso o par  
var estilos = window.getComputedStyle(par);  
  
//depois retornamos a propriedade desejada  
var tam_fonte = estilos.getPropertyValue('font-size');
```

<https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle>

Manipulação de Atributos

- A linguagem JavaScript também fornece métodos para a manipulação de atributos dos elementos, entre eles:
 - `getAttribute("atributo")`
 - Retorna o valor do "atributo", exemplo:
`elem.getAttribute("id");`
`elem.getAttribute("type");`
`elem.getAttribute("placeholder");`
 - Alguns atributos devem ser acessados diretamente, sem utilizar o método `getAttribute`, exemplo: `elem.value`; ou ainda `elem.value="novo valor"`;

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        len = elems.length,  
        callbackExpect = !invert;
```

Manipulação de Atributos

- Outros métodos importantes na manipulação de atributos:
 - `setAttribute(“atributo”, “valor”);`
 - Altera o **valor** do **atributo**, caso o elemento ainda não possua o **atributo**, o **atributo** é inserido com o **valor**.
 - `removeAttribute(“atributo”);`
 - Remove do elemento o **atributo**.

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Coleções de Objetos

- Conforme vimos, o método `document.getElementById()` faz referência a um único objeto.
- Porém, também é possível manusear coleções de objetos do DOM, utilizando, por exemplo:
 - O método `getElementsByTagName("tag")`
 - Retorna todos os elementos de uma determinada *tag*
 - E a propriedade `.children`
 - Retorna os nós filhos de um elemento no DOM

```
.ui-helper-hidden-  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Coleções de Objetos

- Seja o seguinte HTML:

```
<p id="paragrafo-1">Lorem ipsum dolor sit amet</p>
<p class="paragrafo">consectetur adipiscing elit</p>
<p class="paragrafo">Nam fringilla felis et efficitur</p>
<p id="paragrafo-2">Nunc lobortis in eros sed</p>
```

Poderíamos utilizar o seguinte JavaScript para referenciar todos esses elementos:

```
var paragrafos =
document.getElementsByTagName("p");
```

```
.ui-helper-hidden-accessible
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

Coleções de Objetos

- Seja o seguinte HTML:

```
<ul id="lista">
  <li>Lorem ipsum dolor sit amet</li>
  <li>consectetur adipiscing elit</li>
  <li>Nam fringilla felis et efficitur</li>
  <li>Nunc lobortis in eros sed</li>
</ul>
```

Para referenciar todos os itens da lista `#lista` podemos utilizar:

```
var itens =
  document.getElementById("lista").children;
```

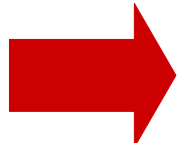
ou ainda:

```
var itens =
  document.getElementById("lista").getElementsByTagName("li");
```

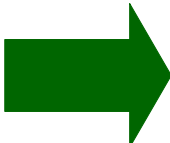
Coleções de Objetos

- Em todos os casos apresentados, as variáveis *paragrafos* e *itens* são arrays e não podem ser “acessadas diretamente”, como fizemos antes:

```
paragrafos.textContent;  
itens.style.backgroundColor="#006600";
```

 **ERRADO!**

```
paragrafos[0].textContent;  
itens[2].style.backgroundColor="#006600";
```

 **CORRETO!**

Coleções de Objetos

- Ou ainda, podemos iterar sobre o array:

```
for (i=0;i<paragrafos.length;i++) {  
    console.log(paragrafos[i].textContent);  
}
```

e

```
for (i=0;i<itens.length;i++) {  
    console.log(itens[i].style.backgroundColor="#006600");  
}
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Coleções de Objetos

- Outros métodos para referenciar os elementos do **DOM**:
 - ***getElementsByClassName('className')***: Retorna um array de objetos de acordo com o nome da classe passada como parâmetro
 - ***querySelector('seletor')***: Retorna o **primeiro** elemento correspondente ao seletor CSS passado como parâmetro.
 - ***querySelectorAll('seletor')***: Retorna um array com todos os elementos correspondentes ao seletor CSS passado como parâmetro.

```
.ui-helper-clearfix {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0 0 0 0;  
position: absolute;  
width: 1px;
```

Eventos

- A linguagem JavaScript fornece um vasto conjunto de recursos para trabalharmos com **eventos**:
 - Os eventos geralmente estão relacionados a **interações do usuário**:
 - Ao clicar em um objeto, ao pressionar uma tecla do teclado, ao passar o mouse sobre um objeto, entre outros.
 - Ou ainda **relacionados ao documento HTML**:
 - Ao carregar página, ao sair da página, ao redimensionar a página, entre outros.

Eventos

- Alguns exemplos de eventos:
 - *onclick*: Disparado quando há um **click do mouse**
 - *onload*: Disparado quando a **página é carregada**
 - *onkeydown*: Disparado quando uma **tecla** do teclado é **pressionada**.
 - *onmouseover*: Disparado quando o **ponteiro do mouse passa sobre** um objeto.
 - *onfocus*: Disparado quando um **campo** de formulário **recebe o foco** (por exemplo o cursor é colocado em um campo de texto)
 - *onfocusout*: Disparado quando um **campo perde o foco**.

Eventos

- Alguns exemplos de eventos:
 - *oninput* / *onchange*: Disparado quando há uma **entrada / alteração de dados em um campo** de formulário.
 - *onsubmit*: Disparado quando um **formulário é submetido** (enviado).

Listas de eventos JavaScript:

http://www.w3schools.com/jsref/dom_obj_event.asp

<https://developer.mozilla.org/en-US/docs/Web/Events>

Manipulação de Eventos

- Podemos manipular eventos diretamente nos elementos HTML por meio de atributos:

```
<span id="span-1" onclick="alert('Fui clicado!');">  
    Lorem ipsum dolor sit amet  
</span>
```

```
<span id="span-2" onmouseover="alert('passaram o mouse!');">  
    Nunc lobortis in eros sed  
</span>
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

Manipulação de Eventos

- Ou ainda, diretamente no JavaScript:

```
<script>
  document.getElementById("span-1")
    .addEventListener("click",function() { alert('fui clicado!'); });
</script>
```

```
<script>
  document.getElementById("span-2")
    .addEventListener("mouseover",function() { alert('passaram o mouse!'); });
</script>
```

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

Manipulação de Eventos

- Observe que:

Utiliza-se uma função anônima

`document.getElementById("span-1").addEventListener("click",function() { alert('fui clicado!'); });`

O prefixo *on* é descartado

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```


Manipulação de Eventos

- Para removermos eventos utilizamos o método `removeEventListener` exemplo:

```
document.getElementById("span-1")  
  .removeEventListener("click", funcaoClick);
```



Importante: não funciona para funções anônimas

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```