



BACHARELADO EM SISTEMAS DE INFORMAÇÃO
CEFET-RJ NOVA FRIBURGO

Programação de Clientes Web

PROF. THIAGO DELGADO PINTO

thiago_dp (at) yahoo (dot) com (dot) br

E s c o p o e m J a v a S c r i p t

versão: 2022.05.10



Licença Creative Commons 4

AGENDA

evolução do EcmaScript

escopo em ES5

modo estrito

hoisting

objeto global

escopo no ES6+

objeto global vs. escopo global

JS	ES	Ano	Descrição
0	-	1995	Criação do JavaScript por Brendan Eich.
1	1	1997	Primeira edição padronizada pela ECMA.
2	2	1998	Mudanças internas somente.
3	3	1999	Adiciona expressões regulares e try/catch.
4	4	-	Nunca foi liberada.
5	5	2009	Adiciona modo estrito, suporte a JSON, String#trim(), Array#isArray() e métodos de iteração em Array.
5.1	5.1	2011	Mudanças internas somente.
6	2015	2015	Adiciona <i>let</i> e <i>const</i> , argumentos com valor <i>default</i> , Array#find(), Array#findIndex()

JS	ES	Ano	Descrição
7	2016	2016	Adiciona operador de potência (**) e método includes() em Array.
8	2017	2017	Adiciona preenchimento (<i>padding</i>) em strings, novas propriedades em Object, <i>async/await</i> , suporte à memória compartilhada.
9	2018	2018	Adiciona suporte à "rest/spread", iteração assíncrona, Promise#finally(), e melhora suporte à exp. regulares.
10	2019	2019	Array#flat(), Array#flatMap(), Array#sort() "estável" (mesma ordem). Muda Object#entries().
11	2020	2020	BigInt (inteiros arbitrários), dynamic import, <i>nullish coalescing op.</i> ("??"), optional chaining ("?."), Promise.allSettled(), String#matchAll(), <i>globalThis</i> , ...
12	2021	2021	Separadores numéricos ("_"), String#replaceAll(), operadores &&=, =, ??= (<i>nullish</i>), Promise.any(), ...

introdução

```
<script>
```

```
  var one = 1;
```

```
  const two = 2;
```

```
</script>
```

```
<script type="module" >
```

```
  var three = 3;
```

```
  const four = 4;
```

```
</script>
```

```
<script type="module" >
```

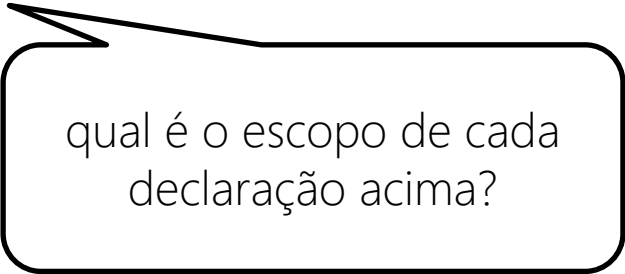
```
  // quais declarações são acessíveis aqui e como???
```

```
</script>
```

escopo em ES5

exemplo 1

```
var one = true;  
function two() {  
  var three = true;  
  function four() { /* ... */ }  
}
```



qual é o escopo de cada
declaração acima?

exemplo 1

```
var one = true; // global
function two() { // global
  var three = true; // função two
  function four() { /* ... */ } // função two
}
```


exemplo 1.1

```
var one = true;

function two() {
  var three = true;
  function four() { /* ... */ }
  if ( true ) {
    var five = true;
    function six() { /* ... */ }
  }
  console.log( five, six );
}
```



e agora ?

exemplo 1.1

```
var one = true; // global
function two() { // global
  var three = true; // função two
  function four() { /* ... */ } // função two
  if ( true ) {
    var five = true; // função two
    function six() { /* ... */ } // função two
  }
  console.log( five, six ); // imprime true, function
}
```

exemplo 2

```
try {  
    throw new Error( '...' );  
} catch ( err ) {  
    // ...  
}  
console.log( err ); // ???
```

exemplo 2

```
try {  
    throw new Error( '...' );  
} catch ( err ) {  
    // ...  
}  
  
console.log( err ); // Reference Error: not defined
```

exemplo 3

```
let obj = {  
  nome: 'Bob',  
  tel: '22334455'  
};
```

```
with ( obj ) {  
  console.log( nome, tel ); // ???  
}
```

exemplo 3

```
let obj = {  
    nome: 'Bob',  
    tel: '22334455'  
};
```

```
with ( obj ) {  
    console.log( nome, tel ); // imprime Bob, 22334455  
}
```

exemplo 3.1

```
'use strict';
```

```
let obj = {  
    nome: 'Bob',  
    tel: '22334455'  
};
```

```
with ( obj ) {  
    console.log( nome, tel ); // ???  
}
```

exemplo 3.1

```
'use strict';
```

```
let obj = {  
    nome: 'Bob',  
    tel: '22334455'  
};
```

```
with ( obj ) {  
    console.log( nome, tel );  
// SyntaxError: Strict mode code may not include a with statement  
}
```


escopo em JavaScript 5

- `global`
- `função`
- `catch`
- `with`, apenas não usando o modo restrito (`'use strict'`)

modo estrito

forma mais *rigorosa* do JavaScript

disponível no ECMAScript 5 ou posterior

evita declarações que dificultam otimização por motores JS

elimina erros silenciosos do JavaScript

lança exceções

proíbe uso de palavras reservadas de versões futuras do JS

intencionalmente tem semânticas diferentes do normal

logo, verifique se seu navegador o suporta (IE < 10 ☺)

pode coexistir, em uma aplicação, junto ao modo normal

pode ser declarado dentro de uma função

não somente em um arquivo

erros de sintaxe

- Uso de octal, como `var n = 023;`
- Uso de `with`
- Uso de `delete` em uma variável, ex.: `delete myVar;`
- Uso de `eval`
- Uso de `arguments` como variável ou parâmetro de função
- Uso de novas palavras reservadas
 - ex. ECMAScript 2015: `implements`, `interface`, `let`, `package`, `private`, `protected`, `public`, `static`, `yield`
- Declaração de funções dentro de blocos
 - ex. `if (a > b) { function f() {} }`

erros em tempo de execução

- variável não declarada
- deletar uma propriedade que não pode ser removida
 - ex., `delete Object.prototype;`
- acessar o **caller** ou o **callee** de **arguments** ou de funções
 - ex., `console.log(arguments.caller);`

hoisting

elevação de declarações (*hoisting*)

Em JavaScript, uma variável (**var**) ou função podem ser declaradas **depois** de elas serem usadas

Isso ocorre porque as declarações são **elevadas** (içadas) automaticamente

Porém, **não** ocorre se a variável for **inicializada**

exemplo 1

```
x = 5;  
console.log( x ); // 5  
var x;
```

exemplo 2

```
console.log( y ); // undefined
```

```
var y = 5; // inicializada
```

exemplo 3

```
z(); // imprime "foo"
```

```
function z() {  
    console.log( 'foo' );  
}
```

exemplo 4

```
var x = 10;
```

```
function y() {  
    console.log( x );  
    var x = 20;  
}
```

```
y(); // ???
```

exemplo 4

```
var x = 10;
```

```
function y() {  
    console.log( 'x =', x );  
    var x = 20;  
}
```

```
y(); // imprime "x = undefined"
```

objeto global

objeto global

aquele que sempre está definido no escopo global

depende do ambiente de execução

em um navegador, é o objeto **window**

em *workers* (ex. Worker, ServiceWorker) é o objeto **self**

no NodeJS, é o objeto **global**

objeto global

atualmente, o objeto **globalThis** foi padronizado

<https://github.com/tc39/proposal-global>

porém, **globalThis** ainda não é suportado por todos os navegadores ou todas as versões do NodeJS

use um *polyfill*, ex. <https://github.com/ungap/global-this>

exemplo no navegador

```
var minhaVar = 10;
```

```
function foo() {  
    console.log( 'foo' );  
}
```

```
console.log( window.minhaVar ); // 10  
window.foo(); // imprime 'foo'
```

exemplo no NodeJS

```
var minhaVar = 10;
```

```
function foo() {  
    console.log( 'foo' );  
}
```

```
console.log( global.minhaVar ); // 10  
global.foo(); // imprime 'foo'
```

exemplo no novo padrão

```
var minhaVar = 10;
```

```
function foo() {  
  console.log( 'foo' );  
}
```

```
console.log( globalThis.minhaVar ); // 10  
globalThis.foo(); // imprime 'foo'
```

escopo de this

se **this** for usado no escopo global, ele aponta para o objeto global (globalThis, window, self ou global)

se **this** for usado dentro de um objeto, ele aponta para o objeto

se **this** for usado dentro de um função...

não usando o modo restrito, ele aponta para a função;

usando o modo restrito, ele é **undefined**

exemplo 1

```
console.log( this === globalThis ); // true
```

exemplo 2

```
var x = 10;  
function y() { return 20; }  
  
console.log( self.x ); // 10  
console.log( self.y() ); // 20
```

exemplo 3

```
function x() {  
    return this;  
}
```

```
console.log( x() === globalThis ); // true  
console.log( typeof x() ); // "object"
```

exemplo 4

```
function y() {  
  'use strict';  
  return this;  
}
```

```
console.log( y() === globalThis ); // false  
console.log( typeof y() ); // "undefined"
```


escopo
no ES6+

exemplo 1

1/2

```
if ( true ) {  
    var one = 1;  
    const two = 2;  
    let three = 3;  
}  
  
console.log( one ); // ???  
console.log( two ); // ???  
console.log( three ); // ???
```

exemplo 1

2/2

```
if ( true ) {  
    var one = 1;  
    const two = 2;  
    let three = 3;  
}  
  
console.log( one ); // 1  
console.log( two ); // two is not defined  
console.log( three ); // three is not defined
```

exemplo 2

1/2

```
{  
  var one = 1;  
  const one = 1;  
}  
  
// Possível ?
```

```
{  
  var one = 1; // SyntaxError: Identifier 'one' has already been declared  
  const one = 1;  
}
```

exemplo 3

1/2

```
<script>
  var one = 1;
  const two = 2;
</script>

<script>
  console.log( one ); // ???
  console.log( self.one ); // ???
  console.log( two ); // ???
  console.log( self.two ); // ???
</script>
```

exemplo 3

2/2

```
<script>
  var one = 1;
  const two = 2;
</script>

<script>
  console.log( one ); // 1
  console.log( self.one ); // 1
  console.log( two ); // 2
  console.log( self.two ); // undefined
  // two fica no escopo global, mas não no objeto global
</script>
```

objeto global vs. escopo global

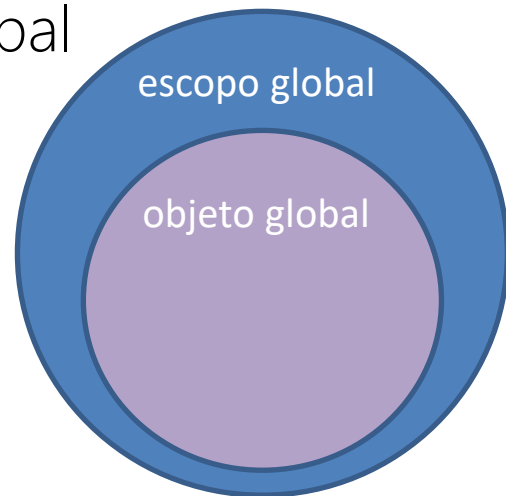
(ao declarar no escopo global...)

uma **var** é incluída no escopo global e no objeto global

let ou **const** são incluídas apenas no escopo global

logo, não são acessíveis pelo objeto global

apenas diretamente



exemplo 4

1/2

```
<script type="module" >
  var one = 1;
  const two = 2;
</script>

<script type="module" >
  console.log( one );           // ???
  console.log( self.one );     // ???
  console.log( two );          // ???
  console.log( self.two );     // ???
</script>
```

exemplo 4

2/2

```
<script type="module" >
```

```
  var one = 1;
```

```
  const two = 2;
```

```
</script>
```

```
<script type="module" >
```

```
  console.log( one );
```

```
// ReferenceError: one is not defined
```

```
  console.log( self.one );
```

```
// undefined
```

```
  console.log( two );
```

```
// ReferenceError: two is not defined
```

```
  console.log( self.two );
```

```
// undefined
```

```
</script>
```

declarações em módulos

são similares à declarações dentro de **funções**

não são adicionadas ao **escopo global** nem ao **objeto global**

exemplo 5 – voltando à introdução

1/3

```
<script >
  var one = 1;
  const two = 2;
</script>
<script type="module" >
  var three = 3;
  const four = 4;
</script>
<script type="module" >
  // quais declarações são acessíveis aqui e como???
</script>
```

exemplo 5 – voltando à introdução

2/3

```
<script >
  var one = 1;
  const two = 2;
</script>
<script type="module" >
  var three = 3;
  const four = 4;
</script>
<script type="module" >
  console.log( one );           // ???
  console.log( self.one );      // ???
  console.log( two );           // ???
  console.log( self.two );      // ???
  console.log( three );         // ???
  console.log( self.three );    // ???
  console.log( four );         // ???
  console.log( self.four );     // ???
</script>
```

exemplo 5 – voltando à introdução

3/3

```
<script >
  var one = 1;
  const two = 2;
</script>
<script type="module" >
  var three = 3;
  const four = 4;
</script>
<script type="module" >
  console.log( one );           // 1
  console.log( self.one );      // 1
  console.log( two );           // 2
  console.log( self.two );      // undefined
  console.log( three );         // Reference Error: three is not defined
  console.log( self.three );    // undefined
  console.log( four );          // Reference Error: four is not defined
  console.log( self.four );     // undefined
</script>
```

escopo em blocos de código

em ES6+, cada bloco de código tem seu próprio escopo

- bloco de código simples, como `{ /*...*/ }`
- `if`, `else if`, `else`, `switch`
- `for`, `while`, `do...while`
- `try`, `catch`
- `<script type="module" >`
- *etc.*

declarações têm visibilidade apenas dentro do bloco

let, **const**, **function**, **class**, *etc.*

exemplo

```
for ( let i = 0; i < 10; ++i ) {  
    // cada loop tem seu próprio ambiente  
}  
console.log( i ); // Reference Error: i is not defined  
  
{  
    const mensagem = 'Olá';  
    console.log( mensagem );  
}  
console.log( mensagem ); // Reference Error: mensagem is not defined
```


referências usadas

W3Schools. **ECMAScript Editions**. Disponível em: https://www.w3schools.com/js/js_versions.asp. Acesso em Agosto de 2019.

Google Chrome Developers. **How Variable Scoping Works in JavaScript**. Publicado 1 de Agosto de 2019. Disponível em: <https://www.youtube.com/watch?v=5LEuJNLfLN0>. Acesso em Agosto de 2019.

MDN. **Strict Mode**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Strict_mode. Acesso em Agosto de 2019.

MDN. **Transitioning to strict mode**. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode/Transitioning_to_strict_mode. Acesso em Agosto de 2019.

Wikipedia. **ECMAScript**. Disponível em: <https://en.wikipedia.org/wiki/ECMAScript>. Acesso em Fevereiro de 2021.



CEFET/RJ - CAMPUS NOVA FRIBURGO, RJ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
bsi.cefet-rj.br

fim

2022.05.10 - Separação dos slide sobre evolução do JavaScript e inclusão do ano 2021.

2021.02.15 - Inclusão de Agenda do conteúdo. Atualização da lista de versões do ECMAScript. Pequenos ajustes visuais.

2019.08.05 - Versão inicial.



Licença Creative Commons 4

ESTE MATERIAL PERTENCE AO PROFESSOR THIAGO DELGADO PINTO
E ESTÁ DISPONÍVEL SOB A LICENÇA CREATIVE COMMONS VERSÃO 4.
AO SE BASEAR EM QUALQUER CONTEÚDO DELE, POR FAVOR, CITE-O.