Entenda a diferença entre var, let e const no JavaScript

Otávio Prado
08/09/2019

Na maioria das linguagens de programação, o escopo das variáveis locais é vinculado ao bloco onde elas são declaradas. Sendo assim, elas “morrem” ao final da instrução em que estão sendo executadas. Será que isso se aplica também à linguagem JavaScript? Vamos verificar:

```
var exibeMensagem = function() {
    var mensagemForaDoIf = 'Caelum';
    if(true) {
        var mensagemDentroDoIf = 'Alura';
        console.log(mensagemDentroDoIf)// Alura ;
    }
    console.log(mensagemForaDoIf); // Caelum

    console.log(mensagemDentroDoIf); // Alura
}
```

Estamos declarando duas variáveis em blocos de código diferentes, qual será o resultado? Vamos testar:

```
exibeMensagem(); // Imprime 'Alura', 'Caelum' e 'Alura'
```

Se `mensagemDentroDoIf` foi declarada dentro do `if`, por que ainda temos acesso a ela fora do bloco desta instrução?

Utilização antes da declaração

Vejamos abaixo outro exemplo de código em JavaScript:

```
var exibeMensagem = function() {
    mensagem = 'Alura';
    console.log(mensagem);
    var mensagem;
}
```

Observe que estamos declarando a variável `mensagem` apenas depois de atribuir um valor e exibí-la no log, será que funciona? Vamos testar!

```
exibeMensagem(); // Imprime 'Alura'
```

Funciona! Como é possível usar a variável `mensagem` antes mesmo de declará-la? Será que o escopo é garantido apenas dentro de onde a variável foi criada?

Hoisting

Em JavaScript, toda variável é **"elevada/içada" (*hoisting*)** até o topo do seu contexto de execução. Esse mecanismo move as variáveis para o topo do seu escopo antes da execução do código.

No nosso exemplo acima, como a variável `mensagemDentroDoIf` está dentro de uma *function*, a declaração da mesma é elevada (*hoisting*) para o topo do seu contexto, ou seja, para o topo da *function*.

É por esse mesmo motivo que "é possível usar uma variável antes dela ter sido declarada": em tempo de execução a variável será elevada (*hoisting*) e tudo funcionará corretamente.

var

Considerando o conceito de hoisting, vamos fazer um pequeno teste usando uma variável declarada com `var` antes mesmo dela ter sido declarada:

```
void function(){
    console.log(mensagem);
}();
var mensagem;
```

No caso da palavra-chave `var`, além da variável ser içada (*hoisting*) ela é automaticamente inicializada com o valor undefined (caso não seja atribuído nenhum outro valor).

Ok, mas qual é o impacto que temos quando fazemos esse tipo de uso?

Imagine que nosso código contenha muitas linhas e que sua complexidade não seja algo tão trivial de compreender.

Às vezes, queremos declarar variáveis que serão utilizadas apenas dentro de um pequeno trecho do nosso código. Ter que lidar com o escopo de função das variáveis declaradas com `var` (escopo abrangente) pode confundir a cabeça até de programadores mais experientes.

Sabendo das "complicações" que as variáveis declaradas com `var` podem causar, o que podemos fazer para evitá-las?

let

Foi pensando em trazer o escopo de bloco (tão conhecido em outras linguagens) que o ECMAScript 6 destinou-se a disponibilizar essa mesma flexibilidade (e uniformidade) para a linguagem.

Através da palavra-chave `let` podemos declarar variáveis com escopo de bloco. Vamos ver:

```
var exibeMensagem = function() {
     if(true) {
          var escopoFuncao = 'Caelum';
          let escopoBloco = 'Alura';

         console.log(escopoBloco); // Alura
     }
     console.log(escopoFuncao); // Caelum
     console.log(escopoBloco);
}
```

Qual será a saída do código acima?

```
exibeMensagem(); // Imprime 'Alura', 'Caelum' e dá um erro
```

Veja que quando tentamos acessar uma variável que foi declarada através da palavra-chave `let` fora do seu escopo, o erro *Uncaught ReferenceError: escopoBloco is not defined* foi apresentado.

Portanto, podemos usar tranquilamente o `let`, pois o escopo de bloco estará garantido.

const

Embora o `let` garanta o escopo, ainda assim, existe a possibilidade de declararmos uma variável com `let` e ela ser *undefined*. Por exemplo:

```
void function(){
    let mensagem;
    console.log(mensagem); // Imprime undefined
}();
```

Supondo que temos uma variável que queremos garantir sua inicialização com um determinado valor, como podemos fazer isso no JavaScript sem causar uma inicialização *default* com *undefined*?

Para termos esse tipo de comportamento em uma variável no JavaScript, podemos declarar constantes por meio da palavra-chave `const`. Vamos dar uma olhada no exemplo:

```
void function(){
    const mensagem = 'Alura';
    console.log(mensagem); // Alura
    mensagem = 'Caelum';
}();
```

O código acima gera um *Uncaught TypeError: Assignment to constant variable*, pois o comportamento fundamental de uma constante é que uma vez atribuído um valor a ela, este não pode ser alterado.

Assim como as variáveis declaradas com a palavra-chave `let`, constantes também tem escopo de bloco.

Além disso, constantes devem ser inicializadas obrigatoriamente no momento de sua declaração. Vejamos alguns exemplos:

```
// constante válida
const idade = 18;

// constante inválida: onde está a inicialização?
const pi;
```

No código acima temos o exemplo de uma constante `idade` sendo declarada e inicializada na mesma linha (constante válida) e um outro exemplo onde o valor não é atribuído na declaração de `pi` (constante inválida) ocasionando o erro *Uncaught SyntaxError: Missing initializer in const declaration.*

É importante utilizar `const` para declarar nossas variáveis, porque assim conseguimos um comportamento mais previsível, já que o valor que elas recebem não podem ser alterado.

Conclusão

Aqui tem um resumo do site constletvar.com:

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

Graças ao *hoisting*, variáveis declaradas com a palavra-chave `var` podem ser utilizadas mesmo antes de sua declaração.

Por outro lado, as variáveis criadas com `let` só podem ser utilizadas após sua declaração, pois, apesar de serem elevadas, elas não são inicializadas.

Além das variáveis declaradas com `var` temos a possibilidade de usar constantes por meio da palavra-chave `const` ou utilizar variáveis com escopo de bloco através da `let`.

Mais JavaScript? Vemos essas e outras características profundas do ECMAScript nos nossos cursos de **JavaScript avançado**. Aqui também um video do Zac Gordon para resumir o que vimos: