ECMAScript e JavaScript

Autor: Jason Lucas Sousa Freitas

02/03/2022

1. Objetivos

Abordar e explanar sobre questões como O que é a ECMA e qual sua diferença do JavaScript, ES2015, strict mode, hoisting, e eventuais subpontos que possam surgir desses tópicos mais gerais.

2. ECMAScript vs JavaScript

2.1 JavaSrcript foi criado em 1995 por Brendam Eich quando trabalhava na Netscape, sendo que o nome original da linguagem era Mocha, que foi mudado para LiveScript antes de receber o nome atual. No ano de 1996, a Netscape decidiu juntar o JS à ECMA International. ECMA (Associação Europeia dos Fabricantes de Computadores) é uma associação que realiza a especificação ou padronização dos sistemas da informação. Ela basicamente define os padrões (ECMA-262) da linguagem e o JS é a implementação desses padrões.

2.2 *Engines* de navegadores interpretam a linguagem de forma distinta, ou seja, algum "entendem" mais a linguagem e tem mais desempenho e outros menos. Caso haja interesse em checar a compatibilidade do ECMAScript com diferentes navegadores, pode ser acessado pelo seguinte link: ECMAScript 6 compatibility table (kangax.github.io). Vale lembrar que essas *engines* não implementam todos os recursos de uma eventual atualização de uma vez, sendo um processo incremental. Diferentes tempos de execução podem compartilhar a mesma *engine* (chrome e node.js por exemplo).

2.3 ES 6 para ES2015. Essa mudança na designação se deu com a ECMA International decidindo mudar o nome dessas atualizações para lançamentos anuais.

2.3.1 Novas *features* da atualização ES2015 pode ser conferida com algumas explicações no site JavaScript ES6 (w3schools.com). Por se tratar de uma grande atualização, apenas alguns pontos serão abordados. Valores de parâmetro padrão foi uma das novas funcionalidades implementadas na linguagem. Consiste em atribuir um valor *default* em determinado parâmetro de determinada função caso este venha como *undefined* no escopo de sua chamada.

2.3.2 *Template strings* (modelos literais) são *strings* que permitem incorporar expressões por meio de um *backtick* (``) no lugar das aspas simples ou duplas e a expressão deve vir encapsulada da seguinte forma: ${expressão}.

```
function template(){
    let stringNormal = 'hello world'; //string convencional
    let stringInterpolada = `${stringNormal}`;
    console.log(stringNormal, ' - ',stringInterpolada);
}
```

Figura 1: Elaborado pelo autor.

2.3.3 *Destructuring* consiste em, como o nome sugere, desestruturar (um objeto ou *array*). Existem diferentes formas de fazer esse procedimento, como por exemplo, a criação de uma variável com mesmo nome da propriedade de um determinado objeto e posteriormente uma atribuição do respectivo objeto. No caso de ser preciso receber mais de uma propriedade desse objeto, será

necessário criar mais variáveis dentro de colchetes, cada uma com o respectivo nome da propriedade que deseja receber e seguir o passo anterior. Abaixo seguem exemplos de outras formas de desestruturação.

```
function destructuring(){
    let array = ['elem1', 'elem2', 'elem3'];
    let [i1, i2, i3] = array;
    console.log(array, i1, i2, i3);
}
```

Figura 2: Elaborado pelo autor.

2.3.4 *Strict mode* define regras a serem seguidas na sintaxe de um algoritmo e na sua execução. Essa notação ajuda a criar códigos mais limpos, evitando por exemplo de variáveis não declaradas de serem usadas ou a exclusão de variáveis(ou objetos). Vale acrescentar que versões mais antigas, tanto de *engines* de navegadores quanto da própria linguagem podem não entender o que é o *strict mode*. Existem diversas outras regras que estarão ilustradas abaixo.

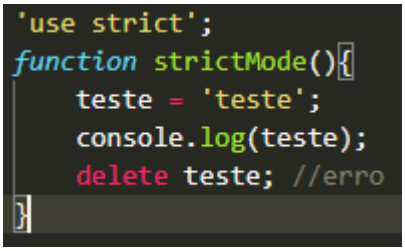

Figura 3: Elaborado pelo autor.

2.3.5 *Hoisting* é um mecanismo do JavaScript que consiste em mover declarações (como variáveis declaradas) para o topo do escopo, sendo local ou global. Sem o *use strict* mostrado anteriormente, uma variável que é inicializada sem ser declarada se torna uma variável global (mesmo que esteja em escopo local) quando o fluxo de execução do código a "encontra". Nos próximos pontos será demonstrado como funciona na prática.

2.3.6 Diferenças entre *let*, *var* e *const*:

2.3.6.1 A notação var para declaração de variáveis possui escopo local ou global assim como as outras variáveis, assim como o *let*, pode modificar a variável mas a primeira diferença está em não poder redeclará-la. O problema no uso do *var* se dá quando for necessário modificar o conteúdo da variável e já tiver uma definição dessa variável antes, que eventualmente pode estar presente em diferentes setores do código.

2.3.6.2 Diferentemente do var e do problema apresentado anteriormente, o *let* tem escopo de bloco (trecho de código cercado por {}), sendo assim, uma variável dentro de um bloco com mesmo nome de uma variável global são variáveis diferentes. Segue abaixo o código acima modificado com *let*.

2.3.6.3 Variáveis declaradas com *const* não permitem alterações nem podem ser declaradas novamente, mas semelhante ao *let*, também possuem escopo de bloco. Objetos declarados com *const* não podem ser alterados mas os valores dos atributos podem. Assim como *var* e *let*, *const* também passa por *hoisting*, porém, devem ser inicializados logo na declaração e não podem ser sobrescritos.

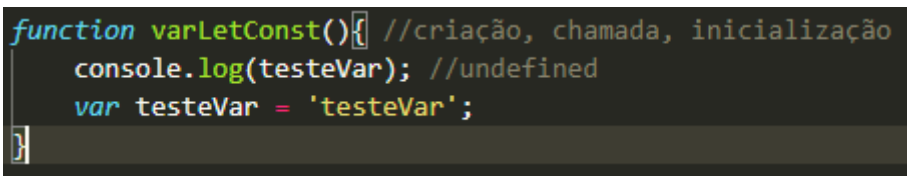

Figura 4: Elaborado pelo autor.

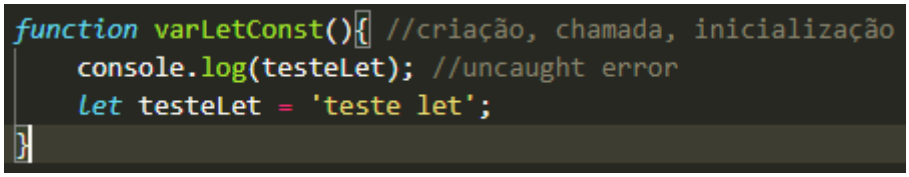

Figura 5: Elaborado pelo autor.

```
function varLetConst(){ //criação, chamada, inicialização
    const testeConst = { 'nome': 'nome', 'idade': 21 };
    console.log(testeConst.nome);
    testeConst.nome = 'Raito Yagami';
    console.log(testeConst.nome);
}
```

Figura 6: Elaborado pelo autor.

2.3.7 Diferenças entre *for*, *forEach* e *for ... of*. Com o *for*, sua definição é a seguinte: “loops através de um bloco de código”, ou seja, semelhante ao *while*, só que um pouco mais completo. Já o *forEach*, por sua vez é usado para iterar em elementos de uma matriz, fazendo isso através de uma função. Por último, o *for ... of* é executado através dos valores de um objeto iterável (*arrays*, *strings*, *maps*, *NodeLists* e outros). Segue abaixo exemplos para cada um deles.

```
function forForEachForOf(){
    let array = [];
    for(let i=0; i < 10; i++){
        array.push(i);
    }
    array.forEach((item) => { // item, index, array
        console.log(item);
    });
}
```

Figura 7: elaborado pelo autor.

```
function forForEachForOf(){
    let array = [];
    let string = 'hello world!';
    let data = { 'name': 'name', 'age': 21 };
    for(let i=0; i < 10; i++){
        array.push(i);
    }
    for(let x of string){
        console.log(x);
    }
    for(let x of array){
        console.log(x);
    }
    for(let x of data){
        console.log(x); //erro
        console.log(data[x]);//erro
    }
}
```

Figura 8: Elaborado pelo autor.

```
function forForEachForOf(){
    let data = { 'name': 'name', 'age': 21 };
    for(let x in data){
        console.log(data[x]);
    }
}
```

Figura 9: Elaborado pelo autor.

3. Bibliografia

1. What's the difference between JavaScript and ECMAScript? (freecodecamp.org)
2. O que é ECMAScript? É o mesmo que JavaScript? - Hcode
3. *https://www.w3schools.com/Js/js_es6.asp*
4. ES6 - Template Literals (Strings de modelo) - ES6 JavaScript - DYclassroom | Divirta-se aprendendo :-)
5. Desestruturação de objetos ES6 - javatpoint
6. JavaScript "use rigoroso" (w3schools.com)
7. | JavaScript Içamento - GeeksforGeeks
8. var, let e const – Qual é a diferença? (freecodecamp.org)
9. JavaScript para de (w3schools.com)
10. Método JavaScript Array forEach() (w3schools.com)
11. JavaScript para Loop (w3schools.com)