02

Comparação entre objetos

Transcrição

Vamos entender o porquê do `indexOf()` nos deixar na mão... No console, executaremos as seguintes instruções:

```
var hoje = new Date();

var n1 = new Negociacao(hoje, 1, 100);
var n2 = new Negociacao(hoje, 1, 100);

n1 == n2 // o resultado é false
```

Usamos `var`, porque não estamos dentro de um bloco. Outros browsers podem reclamar se usarmos a palavra `let`.

O retorno será `false`, porém, `n1` e `n2` são iguais. Ambos tem a mesma data, mesma quantidade e mesmo valor. Nós criamos a variável `n1`, e esta apontará para algo que está na memória. Quando executamos o `new Negociacao`, o JavaScript criará o objeto `Negociacao`, em que teremos a data, a quantidade e o valor. A variável `n1` apontará para o objeto recém criado na memória. `N1` é uma variável de referência que o programa tem para acessar o objeto que está na memória, que criamos com o `new`. Quando criamos o `N2` e damos `new Negociacao`, criaremos um novo objeto em memória do tipo `Negociacao`, que terá a data de hoje, a quantidade e o valor. O `N1` apontará para o objeto na memória. Ou seja, `N1` e `N2` apontará para objetos diferentes. Então, quando digitamos o `n1 == n2`, o que o JavaScript faz é verificar se a variável em `1` está apontando para outra. Como ele não está, o retorno será falso. Por isso, o `indexOf` não funcionou, porque ele também utilizou o `==` por "debaixo dos panos".

Nós temos a lista de negociações que está no nosso modelo, quando o trazemos do Back-end, nós criamos o objeto de novo na memória. Nós recebemos um JSON, e transformamos isto em objetos da memória. Logo, o `indexOf` não está comparando o conteúdo - o valor do objeto - mas sim, se as variáveis estão apontando para outra. Se digitarmos `n1 = n2`, veremos o seguinte retorno:

Agora, `N1` apontará para a mesma negociação de `N2`, ou seja, o mesmo objeto na memória. Ao testarmos, o retorno será `true`, veremos que os dois são iguais.

E o objeto que não é mais acessado, e não tem mais nenhuma variável de referência, será destruído pelo Garbage Colector do JS, liberando memória. A grande questão é que o operador `==` não pode ser usado para comparar objetos.

Mas existe uma peculiaridade... Em JavaScript, sempre trabalhamos com objetos. Veremos isso no exemplo abaixo:

```
var nome1 = 'Flávio';
var nome2 = 'Flávio';
```

```
nome1 == nome2 // o resultado é true
```

Ao compararmos `nome1` com o `nome2`, o retorno será `true`. O mesmo ocorrerá com números...

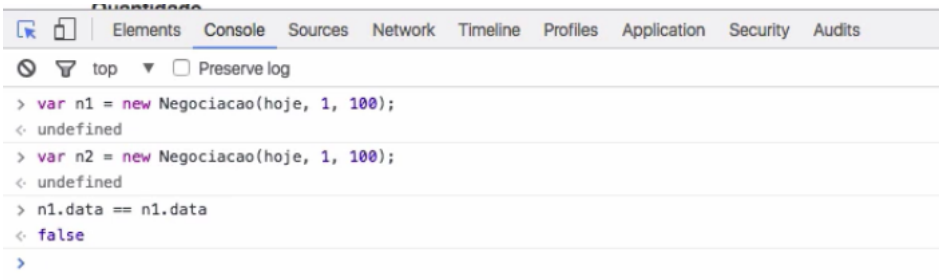
```
var x = 10;
var y = 10;

x == y // o resultado é true
```

Ele dirá que `x` e `y` são igual. Por que isto ocorreu? Quando usamos o `==` com tipos **literais** (ou tipos **primitivos**, em Java), como string, number e boolean, ele não verificará se as variáveis apontam para o mesmo objeto da memória, ele compara se os valores são iguais. Então, nos casos acima, não há problemas... Mas o mesmo não acontecerá com outros tipos de objeto, por exemplo, um `date`. Nós temos dois objetos de memória:

```
var n1 = new Negociacao(hoje, 1, 100);
var n2 = new Negociacao(hoje, 1, 100);
```

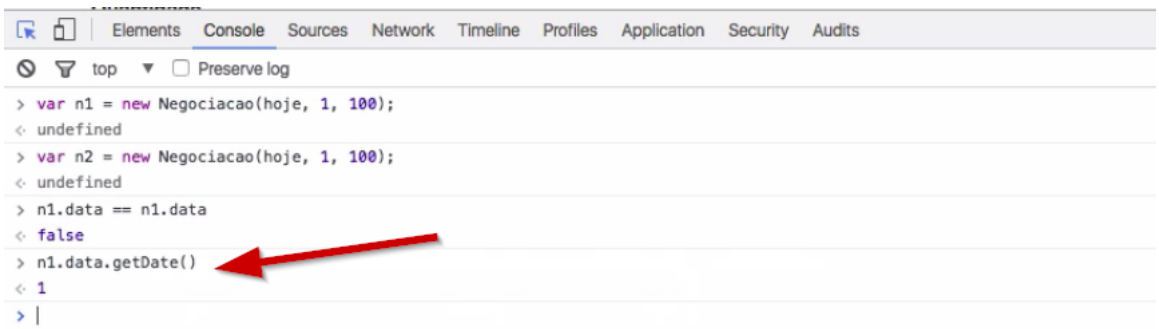
Agora, se digitarmos `n1.data == n2.data`, fique atento! Observe que `n1` e `n2` está recebendo o mesmo objeto `hoje`, o retorno deveria ser `true` ... Porém, não será assim.



Isto ocorreu porque o construtor da nossa classe recebeu a data e cria um novo `date` a partir do `getTime()`. Com isso, um novo objeto será criado na memória e por isso, a comparação não terá o resultado "true". Como vimos anteriormente, com tipos literais, podemos usar `==`. Uma solução é, em vez de compararmos uma variável de referência com outra, podemos comparar as propriedades dos objetos. Faremos o seguinte teste:

```
n1.data == n2.data // o resultado é false
```

Mas o resultado será `false`. No entanto, se fizermos apenas `n1.data.getDate()`, o retorno será um número:



Agora, se fizermos a seguinte comparação:

```
n1.data.getDate() == n2.data.getDate() // o resultado é true
```

O retorno será "verdadeiro". Como `getDate` é um *number*, conseguimos fazer a comparação com o operador `==` .

Em seguida, passaremos a seguinte instrução:

```
n1.data.getDate() == n2.data.getDate() && n1.quantidade == n2.quantidade && n1.valor == n2.valo
```

O retorno será verdadeiro. A nossa comparação não foi feita entre as variáveis, mas sim, entre as duas propriedades diretamente. Como estas eram do tipo literal, a comparação deu certo. No entanto, e se o objeto tivesse 100 propriedades? Fazer a comparação dessa forma seria muito trabalhoso e inviável. Uma estratégia que podemos utilizar é transformar o objeto em uma string. Já vimos que com `JSON.parse` transformamos uma string em um objeto. Usaremos o `JSON.stringify()` :

```
JSON.stringify(n1) == JSON.stringify(n2) // o resultado é true
```

O retorno será verdadeiro. É interessante usar o `stringify()` quando queremos comparar um objeto com outro, no entanto, não será útil se quisermos descobrir se as variáveis apontam para um mesmo objeto. Nós serializamos o objeto, transformando-o em uma string.