03

Usando o método some

Transcrição

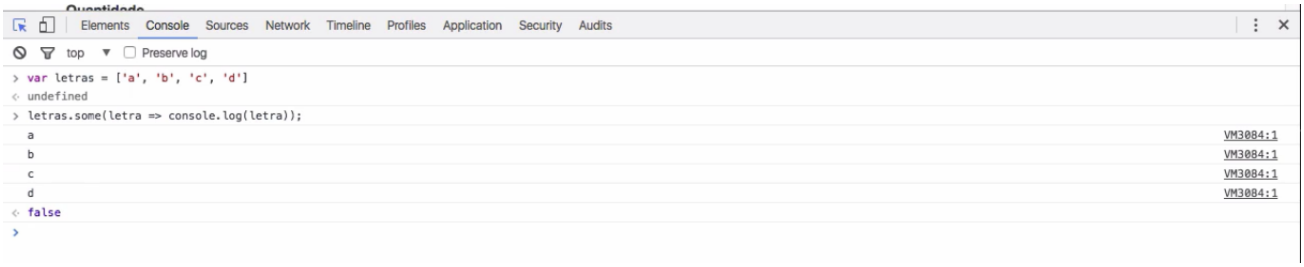
Agora que aprendemos a fazer a comparação, temos um outro problema. Quando voltamos ao nosso código, a função `filter()` receberá "verdadeiro" ou "falso", porque ela vai varrer cada item da lista. Caso o retorno seja `true`, o item será adicionado a lista. Se for `false`, o item será excluído. Antes, poderíamos colocar a instrução dentro do *filter*, considerando que o `indexOf` varria o `listaNegociacoes` procurando a negociação. Porém, não poderemos mais fazer desta forma. Se adicionássemos o `JSON.stringify()`, o código ficaria assim:

```
importaNegociacoes() {

    let service = new NegociacaoService();
    service
        .obterNegociacoes()
        .then(negociacoes =>
            negociacoes.filter(negociacao =>
              JSON.stringify(negociacao) ==

            )
        )
        .then(negociacoes => negociacoes.forEach(negociacao => {
            this._listaNegociacoes.adiciona(negociacao);
            this._mensagem.texto = 'Negociações do período importadas'
        }))
        .catch(erro => this._mensagem.texto = erro);
```

Desta forma, teremos que verificar se a negociação serializada existe dentro de `_listaNegociacoes`, por isso, vamos ter que varrer a lista também. Nosso código começa a ficar mais complicado. Para facilitar, usaremos uma função do ECMAScript 5 pouco usada: a `some()` ... Vamos ver um exemplo:

```
var letras = ['a', 'b', 'c', 'd']
letras.some()
```

Todo array possui a função `some()`, com ela identificamos se o item buscado faz parte da lista, varrendo cada um deles de forma semelhante a um `forEach()`.

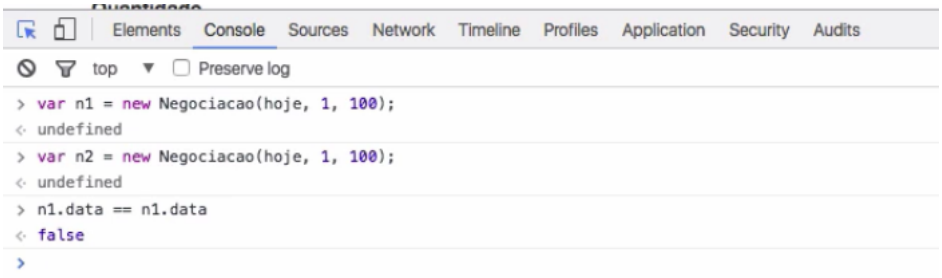


Vamos fazer um teste e buscar pela letra `b`.

```
letras.some(letra => letra == 'b');
```

A função `some()` vai varrer cada item da lista verificando se os elementos são iguais ao critério estabelecido. Enquanto o item for diferente, ele seguirá para o próximo. Quando o elemento for equivalente ao critério `b`, `letras` retornará `true` e não seguirá iterando no array até o fim. Basta encontrar um item que seja correspondente ao critério para que o retorno de `some()` seja "verdadeiro".

No entanto, quando buscamos uma letra que não existe, por exemplo, a letra `e`, o retorno será "falso".

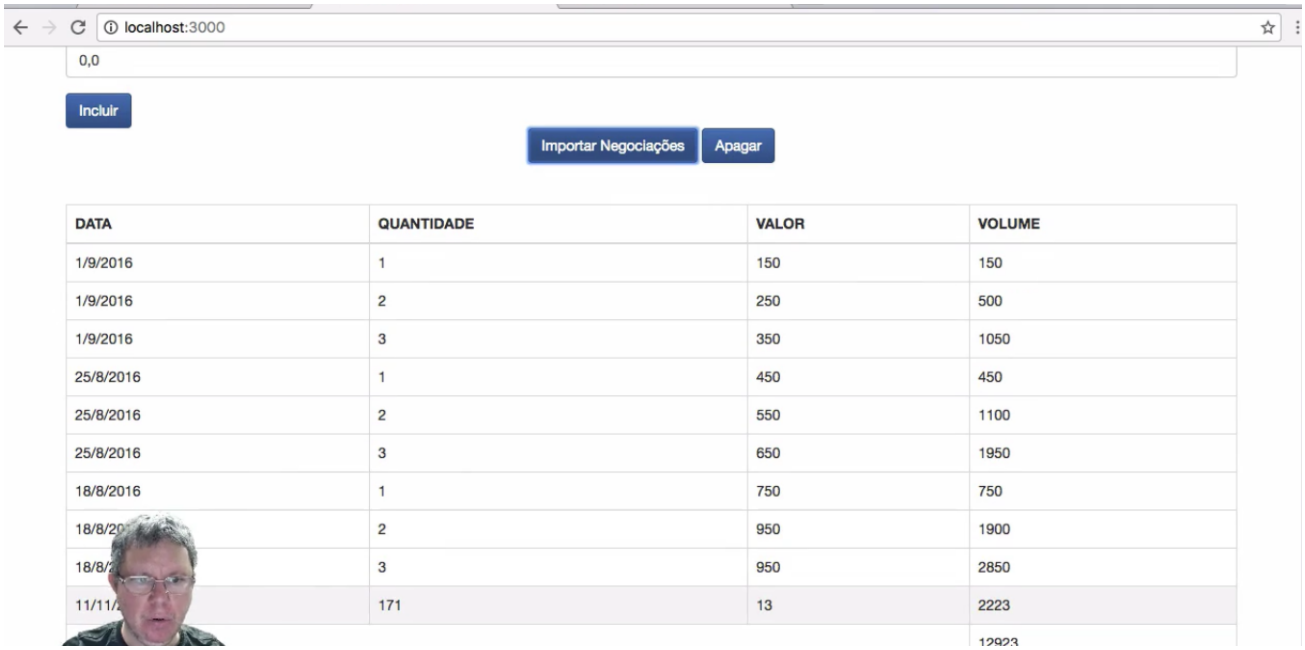


Veremos como podemos usar o `some()` no código.

```
importaNegociacoes() {

    let service = new NegociacaoService();
    service
        .obterNegociacoes()
        .then(negociacoes =>
            negociacoes.filter(negociacao =>
                !this._listaNegociacoes.negociacoes.some(negociacaoExistente =>
                    JSON.stringify(negociacao) == JSON.stringify(negociacaoExistente)))
        )
        .then(negociacoes => negociacoes.forEach(negociacao => {
            this._listaNegociacoes.adiciona(negociacao);
            this._mensagem.texto = 'Negociações do período importadas'
        }))
        .catch(erro => this._mensagem.texto = erro);
```

Estamos fazendo o filtro que varrerá todos os elementos. O primeiro array que será testado (`this._listaNegociacoes`) verificará se cada item já existente é igual a negociação filtrada. Caso seja equivalente, o `some()` retornará "verdadeiro" e o item entrará no filtro de negociações. Como nosso objetivo é que o `some()` retorne "verdadeiro", caso ele siga até o final do array sem encontrar um elemento igual ao filtro, usaremos o sinal de exclamação (`!`). Com isto, no próximo encadeamento da função `then()`, teremos as negociações que não existem dentro de `_listaNegociacoes.negociacoes`.

Se executarmos o código, veremos que quando clicarmos no botão "Importar Negociações", nenhuma negociação será adicionada, porque não foram cadastrados novos dados no formulário. Para fazermos um teste, adicionaremos uma nova negociação e, desta vez, a tabela será alterada.



Apenas a nova negociação foi importada, enquanto as demais foram ignoradas.

Nós conseguimos, misturando os conceitos de Promise, misturando funções do ECMAScript5 (*filter*, *some*...) e usando o artifício de serializar um objeto para saber se ele existe ou não, nós conseguimos filtrar a nossa lista e passando para o próximo encadeamento de `then()` apenas as negociações que não existem de `_listaNegociacoes` . Com essas melhorias, vimos vários conteúdos novos.