07

# Uma solução para método que não altera propriedade

### Transcrição

Nós não vamos nos abdicar de utilizar o Proxy. Mas a primeira tentativa de solucionar o assunto será com uma "gambiarra". Em `ListaNegociacoes`, vamos forçar uma atribuição em `this._negociacoes`.

```
class ListaNegociacoes {

    constructor() {
        this._negociacoes = [];
    }

    adiciona(negociacao) {

        this._negociacoes = [].concat(this._negociacoes, negociacao);
        // this._negociacoes.push(negociacao);
    }
```

Com o `concat()`, podemos passar tanto o *array* quanto o elemento, além de outros parâmetros - e todos serão concatenados. Como forçamos uma atribuição, quando recarregamos a página, veremos no Console:

```
_negociacoes => valor anterior: , novo valor: [object Object]
```

`_negociacoes => valor anterior:` está saindo em branco, porque o `valor anterior` é um array vazio. O `novo valor` tem o `object` porque se trata do novo array.

É possível resolver com esta gambiarra, mas não será esta a solução que utilizaremos. Imagine que fazemos um loop de 100 negociações, se nosso código ficar assim, a cada negociação adicionada, teremos que criar um nova lista e reatribuir para `ListaNegociacoes`. Teríamos problema com a performance. Apesar de termos feito o mesmo no `get`:

```
get negociacoes() {

    return [].concat(this._negociacoes);
}
```

Mas neste caso, quem pede a lista de negociação não fará solicitação infinitas vezes seguidas. É possível varrer a lista diversas vezes, sem chamar `get` sempre. O mesmo não ocorre com `adiciona()`. Se queremos fazer uma operação em lote, usaremos o `adiciona()`. Precisamos descobrir uma outra estratégia - mais de "cangaceiro" do que de "ninja". Ao identificarmos o `adiciona()` como um método no `index.html`, queremos executar a armadilha.

```
<script>

  let lista = new Proxy(new ListaNegociacoes(), {
    set: function(target, prop, value, receiver) {

        console.log(`valor anterior: ${target[prop]} novo valor: ${value}`);
```

```
            return Reflect.set(target, prop, value, receiver);
        }});
        lista.adiciona(new Negociacao(new Negociacao(new Date(), 1, 100));
    </script>
```

Não há uma maneira de interceptarmos o método com o Proxy. Quando chamamos um método no JavaScript, por exemplo o `adiciona()`, ele fará um `get` na função e depois, executará o `Reflect.apply`. Então, será passado o valor para a função dentro do parênteses. Mas o `set` não está incluso no método.

A seguir, resolveremos o assunto.