06

Método que não altera propriedade

Transcrição

Agora que ensaiamos com o `negociacao`, vamos nos focar em criar a Proxy no modelo do `ListaNegociacoes`. No `index.html`, vamos substituir no código `negociacao` por `ListaNegociacoes`. Mudaremos também o `console.log` pelo `lista.adiciona()`.

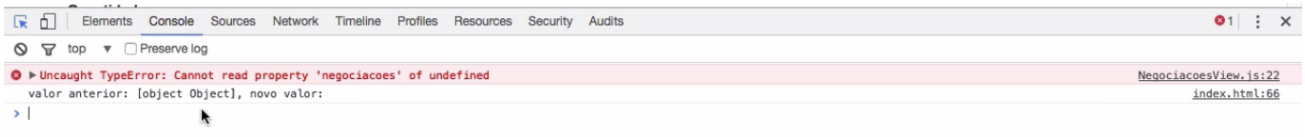
```
<script>

  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 Date(), 1, 100));
</script>
```

Se tentarmos executar o código como está, nada acontecerá. Nós já entendemos que devemos usar o `get` quando estamos olhando uma propriedade e o `set` quando queremos modificar uma propriedade. No entanto, a armadilha não foi disparada no `lista.adiciona()`. Sabe por que isto aconteceu? O `adiciona()` é um método, que no arquivo `ListaNegociacoes.js`, pede internamente para `negociacoes` fazer o `push()`.

```
adiciona(negociacao) {

    this._negociacoes.push(negociacao);

}
```

Porém, nós não atribuímos valores para uma propriedade, sendo assim, o `set` não será chamado. Teremos que encontrar uma solução, porque tanto o método `adiciona()` como o `esvazia()` terá este tipo de problema - ainda que o segundo, atribua um valor para `negociacoes`. Vamos fazer um teste, adicionando o `esvazia()`, no `index.html`:

```
<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 Date(), 1, 100));
  lista.esvazia();
</script>
```



O `esvazia()` executou o código, porque o `valor anterior:` era um objeto e o `novo valor:` ficou vazio. Nós queremos que o interceptador dispare com o `lista.adiciona` . No entanto, por padrão, não conseguiremos fazer isto usando o Proxy do ES6.

Mas não vamos ignorar o que vimos até aqui.