04

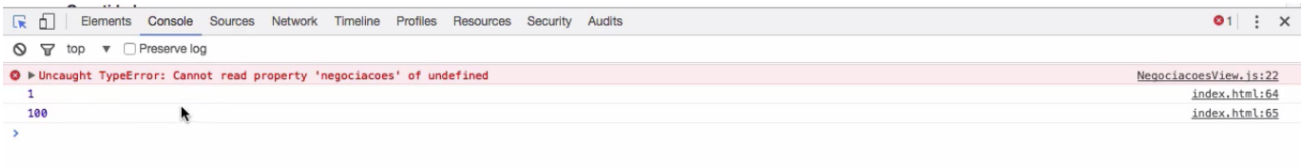
Construindo armadilhas de leitura

Transcrição

Nós já criamos uma Proxy. Em seguida, adicionaremos o `console.log` :

```
<script>
    let negociacao = new Proxy(new Negociacao(new Date(), 1, 100), {});

    console.log(negociacao.quantidade);
    console.log(negociacao.valor);

</script>
```

Se executarmos o código, veremos os valores `1` e `100` impressos no Console.

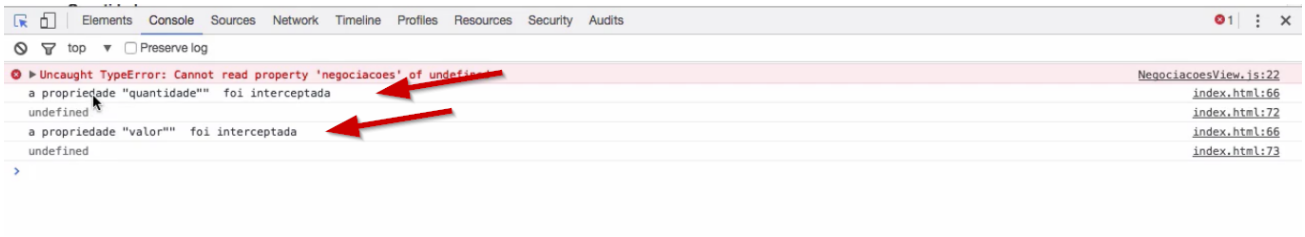


No entanto, queremos executar um código antes de exibirmos o valor de `quantidade` , queremos que seja visualizado um texto informando que a `quantidade` foi acessada. Faremos com que o texto também seja exibido antes da propriedade `valor` . A seguir, adicionaremos uma função que receberá três parâmetros: `target` , `prop` e `receive` .

```
<script>
    let negociacao = new Proxy(new Negociacao(new Date(), 2, 100), {

        get: function(target, prop, receiver) {

              console.log(`a propriedade "${prop}" foi interceptada`);
        }
    });

    console.log(negociacao.quantidade);
    console.log(negociacao.valor);

  </script>
```

O `get` será chamado sempre que tentarmos ler uma das propriedades do objeto. Ao ser chamado, ele tem o `target` (uma referência ao objeto original que está encapsulado pelo Proxy), a propriedade (`prop`) que está sendo acessada, e uma referência (`receiver`) para o Proxy. Agora, antes de ser exibidos os valores das propriedades, os textos devem ser exibidos.

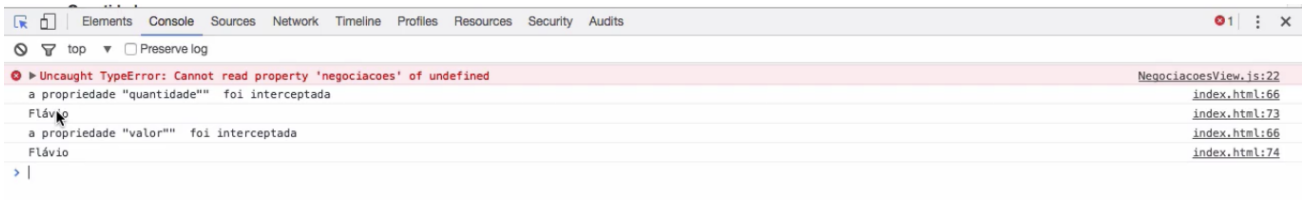
No navegador, vamos obter o resultado esperado:



Porém, os valores das propriedades resultaram em `undefined`. Isto ocorreu, porque quando executamos uma armadilha (*trap*, traduzido para o inglês), é necessário informar qual será o valor retornado após a interceptação da propriedade de leitura. Adicionaremos o `return`:

```
    <script>

        let negociacao = new Proxy(new Negociacao(new Date(), 1, 100), {

            get: function(target, prop, receiver) {

                console.log(`a propriedade "${prop}" foi interceptada`);
                return 'Flávio';
            }
        });
        console.log(negociacao.quantidade);
        console.log(negociacao.valor);
    </script>
```

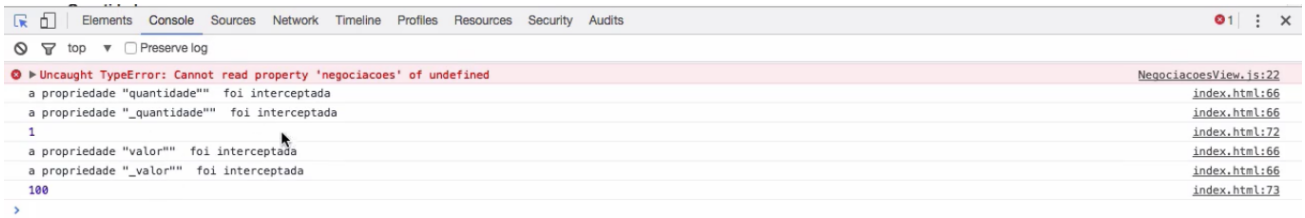
Veja o que será exibido no Console:



Quando acessamos `negociacao.valor`, ele retorna `Flávio`. Mas não é o que queremos... Queremos o valor verdadeiramente guardado. Para isto, vamos pedir auxilio para a API de `Reflect.get()` e os três parâmetros. Nós queremos executar uma operação de leitura.

```
get: function(target, prop, receiver) {

    console.log(`a propriedade "${prop}" foi interceptada`);
    return Reflect.get(target, prop, receiver);
}
```

No navegador, ele imprimirá o texto e os valores das propriedades.



Mas por que a mensagem foi impressa duas vezes, com um pequena diferença. Isto ocorre, porque no arquivo `Negociacao.js` , ele irá interceptar para `quantidade` e `_quantidade` .

```
get quantidade() {

    return this._quantidade;
}

get valor() {
  return this._valor;
}
```

O mesmo acontecerá com `valor` e `_valor` . Então nosso código funciona.

Nós vimos como executar um código, antes da leitura das propriedades do objeto. Mas para resolvermos o problema da atualização automática da View - lembrando que não queremos atualizá-la enquanto estivermos lendo um dado. Mas este não é o nosso foco. O objetivo é encontrar uma forma de executar o código quando uma propriedade é modificada. Veremos mais adiante.