02

E se atualizarmos a view quando o modelo for alterado?

Transcrição

Nós queremos alterar o modelo e chamar a atualização da view automaticamente quando ela for recarregada. Para resolver a questão, imagine que somos caçadores. Nós vamos colocar uma armadilha que será disparada quando alguém pisar no código do `ListaNegociacoes.js` .

Em qual dos métodos seria melhor colocarmos a armadilha, considerando que o nosso objetivo é disparar a atualização da View? A melhor resposta é nos métodos `adiciona()` e `esvazia()` porque são elas que modificam as propriedades da View. Ao invocarmos o método `adiciona()` simultaneamente chamaremos uma armadilha que atualizará a View.

Em seguida, adicionaremos a `armadilha` como parâmetro de `constructor()` , também vamos incluir uma propriedade chamada `_armadilha` .

```
class ListaNegociacoes {

    constructor(armadilha) {

        this._negociacoes = [];
        this._armadilha = armadilha;
    }
//...
```

A `armadilha` é uma função, que guardaremos no `constructor()` para chamá-la posteriormente - ou seja, quando chamarmos o `adiciona()` e `esvazia()` .

Em `NegociacaoController` , vamos adicionar o `function()` dentro de `new ListaNegociacao` . Após passarmos uma função anônima como parâmetro, ela vai executar a seguinte linha de código:

```
this._negociacoesView.update(this._listaNegociacoes);
```

O trecho do código ficará assim:

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');
        this._listaNegociacoes = new ListaNegociacoes(function() {

            this._negociacoesView.update(this._listaNegociacoes);
        });
       //...
```

Fique tranquilo sobre o `NegociacoesView` ser declarado depois. O trecho acima só será executado quando os métodos `adiciona()` e `esvazia()` forem chamados.

Vamos manter o `negociacoesView.update()` mais abaixo, para que seja feita a primeira renderização da lista. Porém, vamos apagar a mesma linha do `adiciona()` e `apaga()`.

```
 adiciona(event) {

    event.preventDefault();

    this._listaNegociacoes.adiciona(this._criaNegociacao());
    this._mensagem.texto = 'Negociação adicionada com sucesso';
    this._mensagemView.update(this._mensagem);

    this._limpaFormulario();
}

// código posterior omitido

apaga() {

    this._listaNegociacoes.esvazia();
    // Linha abaixo comentada, não precisamos mais dela
    // this._negociacoesView.update(this._listaNegociacoes);

    this._mensagem.texto = "Negociações removidas com sucesso";
    this._mensagemView.update(this._mensagem);
}
```

Depois, faremos alterações no `ListaNegociacoes.js`:

```
adiciona(negociacao) {

    this._negociacoes.push(negociacao);
    this._armadilha(this);

}
```

No `this.armadilha()` passamos o *model* como parâmetro, que será acessado com o `this`. No `NegociacaoController.js`, adicionaremos o `model` em `_listaNegociacoes` e também no `update()`.

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');
        this._listaNegociacoes = new ListaNegociacoes(function(model) {
```

```
            this._negociacoesView.update(model);
        });
```

Criamos o `_listaNegociacoes`, passei a função que será chamada quando usarmos o `adiciona()` e o `esvazia()`, os dois métodos passaram o modelo como parâmetro. Depois, voltaremos para o `ListaNegociacoes.js`, e adicionaremos o `this.armadilha(this)` no método `esvazia()`:

```
esvazia() {

    this._negociacoes = [];
    this._armadilha(this);
}
```

Vamos recarregar a página e testar se nossa armadilha funcionará. Mas ela não vai... No Console, veremos uma mensagem de erro:

Ele nos diz que o `this.armadilha` não é uma função dentro de `ListaNegociacoes`.

Teremos que fazer alguns ajustes, primeiramente, adicionaremos o prefixo `_` ao `armadilha`:

```
class ListaNegociacoes {

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

    adiciona(negociacao) {
        this._negociacoes.push(negociacao);
        this._armadilha(this);
    }

    get negociacoes() {
        return [].concat(this._negociacoes);
    }

    esvazia() {
        this._negociacoes = [];
        this._armadilha(this);
    }
}
```

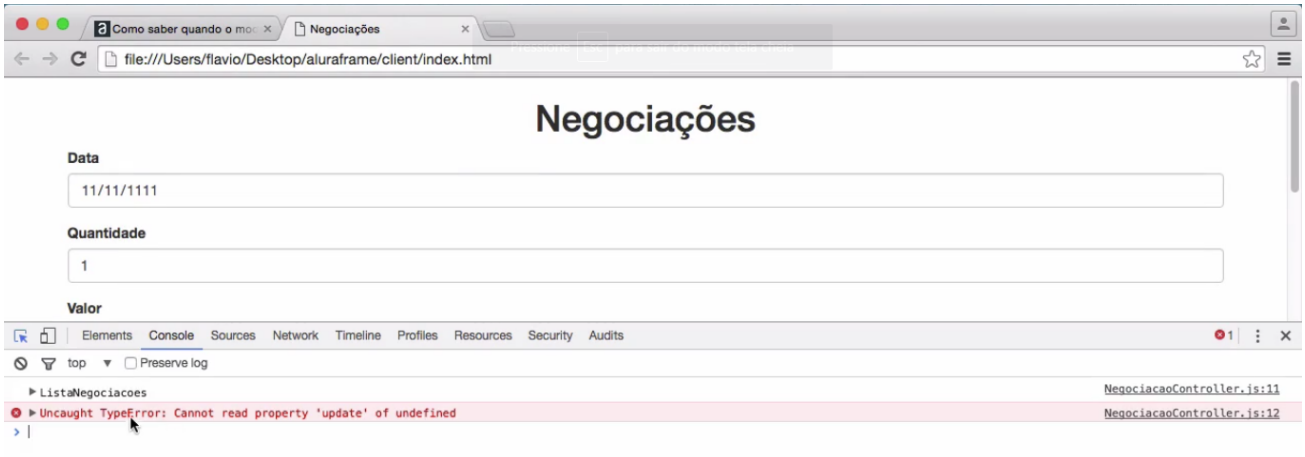
Depois, temos que resolver um problema no `constructor` de `NegociacaoController()`.

```
this._listaNegociacoes = new ListaNegociacoes(function(model) {
            this._negociacoesView.update(model);
```

Quando o `_armadilha` é executado, o `this._negociacoesView` não existe. Isto ocorre, porque a função é executada em um contexto dinâmico de `ListaNegociacoes()`. O `this` dentro de uma função, para ser avaliado, depende do contexto no qual ela foi executada - no caso, o contexto será de `ListaNegociacoes`. Então, `this` é a `_listaNegociacoes`, porém, esta não tem `_negociacoesView`. Para resolver, precisamos que o `this` seja `NegociacaoController`, porque toda função JavaScript tem o escopo `this` dinâmico, que varia de acordo com o contexto. Vamos fazer um teste, adicionando o `console.log()` em `_listaNegociacoes`:

```
this._listaNegociacoes = new ListaNegociacoes(function(model) {
            console.log(this);
            this._negociacoesView.update(model);
```

Testaremos preencher o formulário, no Console, veremos outra mensagem de erro:



Ele mostra que `this` é o `ListaNegociacoes`. É assim, porque a função está sendo executada no contexto de `_listaNegociacoes`. Tem como fazer com o contexto de `this` seja o `NegociacaoController`? É o que veremos mais adiante.