04

Parâmetros REST

Transcrição

Nós queremos simplificar ainda mais o nosso código. Nós estamos passando as propriedades `adiciona` , `esvazia` e `texto` dentro de um array:

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');

        this._negociacoesView = new NegociacoesView($('#negociacoesView'));

        this._listaNegociacoes = new Bind (
                new ListaNegociacoes(),
                this._negociacoesView($('#negociacoesView')),
                ['adiciona', 'esvazia']);


        this._mensagemView = new MensagemView($('#mensagemView')),
        this._mensagem = new Bind(
            new Mensagem(),
            this._mensagemView,
            ['texto']);
    }
//...
```

Vamos retirar as propriedades do array.

```
this._listaNegociacoes = new Bind (
        new ListaNegociacoes(),
        this._negociacoesView($('#negociacoesView')),
        'adiciona', 'esvazia');
```

Mas, então, teremos uma `ListaNegociacoes` com uma View e dois parâmetros? Na classe `Bind` , veremos que são aceitos apenas três parâmetros.

```
class Bind {

    constructor(model, view, props) {

        let proxy = ProxyFactory.create(model, props, model =>
            view.update(model));

        view.update(model);
```

```
            return proxy;
        }
    }
```

No entanto, a `ProxyFactory` precisa receber um array, por isso, as propriedades eram passadas entre `[]` (colchetes). Contudo, quando o último parâmetro de um construtor, função ou método é variável, podemos usar o parâmetro **REST** operator (`...`):

```
constructor(model, view, ...props) {

    let proxy = ProxyFactory.create(model, props, model => {
        view.update(model)
    });

    view.update(model);
    return proxy;
}
```

Vamos entender o que será feito, relembrando o `Bind` do `NegociacaoController.js` :

- O primeiro parâmetro recebido pelo `Bind` é o `model` , o segundo é a `view` , e a partir do terceiro, eles caem dentro do `...props` - podendo ser diversos, como um array. No nosso caso, `...props` é um array com duas posições (`adiciona` e `esvazia`). É isso que o `create()` do `ProxyFactory` espera receber. Com a pequena adição do REST. Isto também nos permite fazer uma associação com apenas um parâmetro, como no caso do `texto` de `_mensagem` , sem colocá-lo em um array.

O rest operator (`...`) não deve ser adicionado no primeiro parâmetro, porque isso traria problemas para os seguintes. Por exemplo, se fizéssemos `...model` :

constructor (...model, view, ...props){ }

Para gerarmos um array com *n* parâmetros, devemos usar o REST apenas no último, outro vantajoso recurso do ECMAScript.

Tem mais uma melhoria que podemos fazer no código... No início da aplicação, criamos uma instância de `NegociacoesView` , que era guardada na propriedade do `_negociacoesView` :

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');

        this._negociacoesView = new NegociacoesView($('#negociacoesView'));
//...
```

Também criamos o `MensagemView()` e guardamos na propriedade da `controller`. Mas se observarmos o código, veremos que não utilizamos a View novamente em `NegociacoesController`. Quem manipulará a View será a nossa associação, que mediante a atualização do modelo, recarregará a View. Então, não precisamos ter as propriedades `this._negociacoesView` e `this._mensagemView`, afinal, elas não são utilizadas. Passaremos as instâncias da View de forma mais direta:

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');

        this._listaNegociacoes = new Bind(
            new ListaNegociacoes(),
            new NegociacoesView($('#negociacoesView')),
            'adiciona', 'esvazia')

        this._mensagem = new Bind(
            new Mensagem(), new MensagemView($('#mensagemView')),
            'texto');
    }
```

Não vamos mais manipular a View manualmente, ela será atualizada automaticamente quando o modelo for alterado. Mas precisaremos de um modelo para trabalhar. Quando usamos o `new Bind()`, associaremos o modelo com a View e o `ListaNegociacoes` será o Proxy. Então, agora, o código ficou mais enxuto.