03

De namespaces para ES2015 modules

Transcrição

A sintaxe de módulos do ES2015 considera cada script um módulo e através das instruções `import` e `export` importamos e exportamos artefatos respectivamente.

Adequando todos os arquivos para o sistema de módulos do ES2015:

```
export abstract class View<T> {

    protected _elemento: JQuery;

    constructor(seletor: string) {

        this._elemento = $(seletor);
    }

    update(model: T) {

        this._elemento.html(this.template(model));
    }

    abstract template(model: T): string;

}
```

```
import { View } from './View';
import { Negociacoes } from '../models/Negociacoes';

export class NegociacoesView extends View<Negociacoes> {

    template(model: Negociacoes): string {

        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                    <th>VOLUME</th>
                </tr>
            </thead>

            <tbody>
                ${model.paraArray().map(negociacao =>
                    `
                        <tr>
                            <td>${negociacao.data.getDate()}/${negociacao.data.getMonth() +1}/$
                            <td>${negociacao.quantidade}</td>
                            <td>${negociacao.valor}</td>
```

```
                        <td>${negociacao.volume}</td>
                    <tr>
                `
            ).join('')}
        </tbody>

        <tfoot>
        </tfoot>
    </table>
    `;
  }
}
```

```
import { View } from './View';

export class MensagemView extends View<string> {

    template(model: string): string {

        return `<p class="alert alert-info">${model}</p>`;
    }

}


import { Negociacao } from './Negociacao';

export class Negociacoes {

    private _negociacoes: Negociacao[] = [];

    adiciona(negociacao: Negociacao): void {

        this._negociacoes.push(negociacao);
    }

    paraArray(): Negociacao[] {

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


export class Negociacao {

    constructor(private _data: Date, private _quantidade: number, private _valor: number) {}

    get data() {

        return this._data;
    }

    get quantidade() {

        return this._quantidade;
```

```
    }

    get valor() {

        return this._valor;
    }

    get volume() {

        return this._quantidade * this._valor;
    }
}
```

```
import { NegociacoesView } from '../views/NegociacoesView';
import { MensagemView } from '../views/MensagemView';
import { Negociacoes } from '../models/Negociacoes';
import { Negociacao } from '../models/Negociacao';

export class NegociacaoController {

    private _inputData: JQuery;
    private _inputQuantidade: JQuery;
    private _inputValor: JQuery;
    private _negociacoes = new Negociacoes();
    private _negociacoesView = new NegociacoesView('#negociacoesView');
    private _mensagemView = new MensagemView('#mensagemView');

    constructor() {
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');
        this._negociacoesView.update(this._negociacoes);
    }

    adiciona(event: Event) {

        event.preventDefault();

        const negociacao = new Negociacao(
            new Date(this._inputData.val().replace(/-/g, ',')),
            parseInt(this._inputQuantidade.val()),
            parseFloat(this._inputValor.val())
        );

        this._negociacoes.adiciona(negociacao);

        this._negociacoesView.update(this._negociacoes);
        this._mensagemView.update('Negociação adicionada com sucesso!');
    }
}
```

```
import { NegociacaoController } from './controllers/NegociacaoController';

const controller = new NegociacaoController();
$('.form').submit(controller.adiciona.bind(controller));
```

Isso ainda não é suficiente, precisamos utilizar um loader, uma biblioteca que seja capaz de carregar o módulo `app.js` e a partir deles carregar todos os demais módulos.