02

ES2015 e módulos

Transcrição

Vamos precisar seguir uma série de etapas para usar o sistema de módulos do ECMAScript 2015. Mas a primeira coisa que faremos - e ainda será insuficiente - será converter todos os scripts feitos para o sistema.

Começaremos pela classe `View.js`, que é dependência da classe `MensagemView`. O que faremos? No módulo do ES6, todos os scripts também serão. Isto significa que todo o conteúdo dentro do arquivo JS não será acessível para outros e não estará no escopo global. Se tivéssemos ativado o sistema de módulo do ES 6, nenhuma definição de classe estaria no escopo global e a aplicação não funcionaria. Por exemplo, se `MensagemView` quiser utilizar a `View`, teremos que importar a classe `View` para o módulo `View.js`:

```
export class View{

    constructor(elemento) {

      this._elemento = elemento;

    }

    template() {

        throw new Error('O método template deve ser implementado');

    }

    update(model) {

        this._elemento.innerHTML = this.template(model);
    }
}
```

Se incluirmos a palavra `export` na classe, quando o `MensagemView.js` tentar utilizar a `View`, ele não conseguirá fazer a importação. Faremos isto a seguir:

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

export class MensagemView extends View {

    constructor(elemento) {

        super(elemento);
    }

    template(model) {

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

Nós explicitamos que queremos importar algo do módulo `View`. Relembrando, cada script é considerado um módulo por padrão e todo conteúdo não cairá no escopo global. Como `View` está na mesma pasta que `MensagemView`, usamos apenas o `./`, sem precisar especificar a extensão no fim. Também, especificamos dentro das chaves o que queremos importar (`View`). Observe que também exportamos o `MensagemView`, porque ela será importada pelo `NegociacaoController.js`.

Repetiremos o processo de importação e exportação em `NegociacaoView.js`:

```
import {View} from './View';
export class NegocicaoView extends View {

    constructor(elemento) {

        super(elemento);
    }
//...
```

E depois, adicionaremos o `export` em `DateHelper.js`:

```
export class DateHelper {

    constructor() {

        throw new Error('Esta classe não pode ser instaciada');
    }
//...
```

Outros arquivos precisam da classe `HttpService`, iremos exportá-la:

```
export class HttpService {

    _handleErrors(res) {
        if(!res.ok) throw new Error(res.statusText);
        return res;
    }
//...
```

Faremos o mesmo em `NegociacaoService.js`. Observe que também importaremos o `HttpService`:

```
import {HttpService} from './HttpService';
 export class NegociacaoService {

      constructor(){

          this._http = new HttpService()
      }
  //...
```

`NegociacaoService` depende da definição de classe de `HttpService` que estamos importando. A próxima alteração será em `ProxyFactory.js`:

```
export class ProxyFactory {

    static create(objeto, props, acao) {
        return new Proxy(objeto, {

            get(target, prop, receiver) {

                if(props.includes(prop) && ProxyFactory._ehFuncao(target[prop])) {

                  console.log(`interceptando ${prop}`);
                  let retorno = Reflect.apply(target[prop], target, arguments);
                  acao(target);
                  return retorno;
                }
            }
        }
//...
```