08

Interface de função

Transcrição

O método `importaNegociacoes` de `NegociacaoServive` aceita receber uma função. Não é à toa que usamos o tipo `Function`. No entanto, podemos passar qualquer função. Por exemplo, uma que não recebe um Response e muito menos o retorna:

Alguém nos impede de fazermos:

```
// dará erro, passar function() {} ou () => {}
 this._service
     .obterNegociacoes(function() {})
     .then(negociacoes => {
         negociacoes.forEach(negociacao =>
             this._negociacoes.adiciona(negociacao));
         this._negociacoesView.update(this._negociacoes);
     });
```

Podemos restringir um pouco mais quais funções podem ser passadas para o método através de uma interface. Com ela, apenas as funções que receberem o único parâmetro Response e retornarem Response serão válidos:

```
// app/ts/services/NegociacaoService.ts

import { NegociacaoParcial, Negociacao } from '../models/index';

export class NegociacaoService {

    obterNegociacoes(handler: ResponseHandler): Promise<Negociacao[]> {

        return fetch('http://localhost:8080/dados')
            .then(res => handler(res))
            .then(res => res.json())
            .then((dados: NegociacaoParcial[]) =>
                dados.map(dado => new Negociacao(new Date(), dado.vezes, dado.montante))
            )
            .catch(err => console.log(err));

    }
}

export interface ResponseHandler {

    (res: Response): Response
}
```

A interface `ResponseHandler` é o tipo que define que a função deve receber um parâmetro do tipo `Response` e devolver um `Response`. Inclusive, lá no método `obterNegociacoes` mudamos o tipo de `Function` para `ResponseHandler`.

Agora, em `NegociacaoController`, nosso código não compila com a função indevida que passarmos e somos alertados disso através do compilador do TypeScript. Se passarmos `isOk` novamente, tudo continuará funcionando, pois `isOk` segue a assinatura da interface.

Podemos até fazermos isso se desejarmos:

```
// app/ts/controllers/NegociacaoController.ts

// código anterior omitido
    @throttle()
    importaDados() {

        const isOk: ResponseHandler = (res: Response) => {
            if(res.ok) return res;
            throw new Error(res.statusText);
        }

        this._service
            .obterNegociacoes(isOk)
            .then(negociacoes => {
                negociacoes.forEach(negociacao =>
                    this._negociacoes.adiciona(negociacao));
                this._negociacoesView.update(this._negociacoes);
            });

    }
// código posterior omitido
```

Ou, para evitarmos que digitar um pouco, podemos passar uma arrow function diretamente:

```
// app/ts/controllers/NegociacaoController.ts

// código anterior omitido

    @throttle()
    importaDados() {

        this._service
            .obterNegociacoes(res => {
                if(res.ok) return res;
                throw new Error(res.statusText);
            })
            .then(negociacoes => {
                negociacoes.forEach(negociacao =>
                    this._negociacoes.adiciona(negociacao));
                this._negociacoesView.update(this._negociacoes);
            });

    }
// código posterior omitido
```

Nesse caso nem precisamos mais importar `ResponseHandler`.