05

Revisando decorators

Transcrição

Do jeito que nossa aplicação esta, se clicarmos muitas vezes em seguida no botão de importar negociações executaremos várias requisições para o servidor. Não há necessidade de tanta ansiedade por parte do usuário, mas para nos prevenirmos disso, podemos ignorar cliques que estejam dentro de uma janela de tempo definida pelo programador. No caso, se o usuário clicar em menos de meio segundo, cancelaremos a ação dele e iniciaremos uma nova. Em suma, essa solução utilizará um timer para processar as ações do usuário:

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

import { NegociacoesView, MensagemView } from '../views/index';
import { Negociacao, Negociacoes } from '../models/index';
import { domInject } from '../helpers/decorators/index';
import { NegociacaoParcial } from '../models/index';

let timer = 0;

export class NegociacaoController {

   // código posterior omitido

    importaDados() {

        function isOk(res: Response) {

            if(res.ok) {
                return res;
            } else {
                throw new Error(res.statusText);
            }
        }

        clearTimeout(timer)
        timer = setTimeout(() => {

            fetch('http://localhost:8080/dados')
                .then(res => isOk(res))
                .then(res => res.json())
                .then((dados: NegociacaoParcial[]) => {
                    dados
                        .map(dado => new Negociacao(new Date(), dado.vezes, dado.montante))
                        .forEach(negociacao => this._negociacoes.adiciona(negociacao))
                    this._negociacoesView.update(this._negociacoes);
                })
                .catch(err => console.log(err));
        }, 500);

    }
}
// código posterior omitido
```

Excelente, nosso código funciona. Quando o usuário clicar pela primeira vez, qualquer temporizador que esteja sendo processado será cancelado e um novo será criado. É esse temporizador que executará nosso código. Se os meio segundos passarem, o temporizador executará nosso código. Então, se ele ficar clicando várias vezes em menos de meio segundo, apenas a última operação será executada.

No entanto esse código é muito invasivo, pois mistura sua lógica com a lógica do método do controller. Além disso, podemos querer usar a mesma estratégia em outros métodos da nossa aplicação. Aprendemos a criar decorators que isolam um código e que podem ser utilizados em vários lugares. Vamos criar o decorator chamado `throttle`, aliás, o padrão de projeto que lida com o problema que estamos enfrentando:

```
// app/ts/helpers/decorators/throttle.ts

export function throttle(milissegundos = 500) {

    return function(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

        const metodoOriginal = descriptor.value;

        let timer = 0;

        descriptor.value = function(...args: any[]) {

            clearInterval(timer);
            timer = setTimeout(() => metodoOriginal.apply(this, args), milissegundos);
        }

        return descriptor;
    }
}


// app/ts/helpers/decorators/index.ts

export * from './logarTempoDeExecucao';
export * from './domInject';
export * from './throttle';
```

Agora, basta utilizarmos nosso decorator no método do nosso controller:

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

import { NegociacoesView, MensagemView } from '../views/index';
import { Negociacao, Negociacoes } from '../models/index';
import { domInject, throttle } from '../helpers/decorators/index';
import { NegociacaoParcial } from '../models/index';

// não precisa mais da variável time!

export class NegociacaoController {

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

```
        function isOk(res: Response) {

            if (res.ok) {
                return res;
            } else {
                throw new Error(res.statusText);
            }
        }

        fetch('http://localhost:8080/dados')
            .then(res => isOk(res))
            .then(res => res.json())
            .then((dados: NegociacaoParcial[]) => {
                dados
                    .map(dado => new Negociacao(new Date(), dado.vezes, dado.montante))
                    .forEach(negociacao => this._negociacoes.adiciona(negociacao))
                this._negociacoesView.update(this._negociacoes);
            })
            .catch(err => console.log(err));


    }
}
// código posterior omitido
```

Contudo, se usarmos o mesmo decorator como método `adiciona()` de `NegociacaoController` teremos um problema. A página será recarregada. Isso acontece, porque como postergarmos a execução do nosso código, postergamos também `event.preventDefault()`. A boa notícia, é que podemos acessar `event` implicitamente dentro da chamada de qualquer função, sem necessariamente ele ter sido passado como parâmetro ou não para ela. Nesse sentido, vamos realizar sempre um `event.preventDefault()` em todos os métodos que usarem nosso decorator `throttle`:

```
// app/ts/helpers/decorators/throttle.ts

export function throttle(milissegundos = 500) {

    return function(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

        const metodoOriginal = descriptor.value;

        let timer = 0;

        descriptor.value = function(...args: any[]) {
            if(event) event.preventDefault();
            clearInterval(timer);
            timer = setTimeout(() => metodoOriginal.apply(this, args), milissegundos);
        }

        return descriptor;
    }
}
```

Veja também que o TypeScript sabe que event é uma variável implícita que pode existir ou não em uma função e por isso adota o tipo implícito `Event`, fantástico!