10

A classe de serviço

Transcrição

Vamos dar uma olhadinha na seguinte chamada da função `fetch`:

```
fetch('http://localhost:3000/notas')
```

Teremos esse código espalhado em todos os locais da nossa aplicação que precisarem buscar negociações e se o endereço mudar, teremos que alterar em todos os lugares. Além disso, esse techo de código por si só não deixa clara sua intenção.

Podemos isolar o acesso à API em uma classe de serviço. Em todos os lugares que precisamos interagir com a API faremos através desta classe.

Vamos criar a pasta e o módulo `app/nota/service.js`. Quando ele por importado, devolverá sempre um objeto que representa o serviço. Aliás, vamos isolar a chamada de `handleStatus` e tratar qualquer erro de baixo nível que aconteça lançando em seu lugar uma mensagem de alto nível:

```
import { handleStatus } from '../utils/promise-helpers.js';

const API = `http://localhost:3000/notas`;

export const notasService = {
    listAll() {
        return fetch(API)
        // lida com o status da requisição
        .then(handleStatus)
        .catch(err => {
            // a responsável pelo logo é do serviço
            console.log(err);
            // retorna uma mensagem de alto nível
            return Promise.reject('Não foi possível obter as notas fiscais');
        });


    }
};
```

Não há necessidade de utilizarmos uma classe aqui porque não queremos criar instâncias de uma classe. Nosso serviço não guardará estado, a não ser a URL da API que ficará no escopo do módulo e, através de closure, o objeto retornado terá acesso à variável `API`.

Agora, vamos utilizá-lo em `app/app.js`:

```
// app/app.js
import { log } from './utils/promise-helpers.js';
import './utils/array-helpers.js';
// importa dando um apelido para o artefato importado
import { notasService as service } from './nota/service.js';
```

```
const sumItems = code => notas =>
    notas.$flatMap(nota => nota.itens)
    .filter(item => item.codigo == code)
    .reduce((total, item) => total + item.valor, 0)

document
.querySelector('#myButton')
.onclick = () =>
    // utilizando o serviço
    service
    .listAll()
    .then(sumItems('2143'))
    .then(console.log)
    .catch(console.log);
```

Uma coisa diferente que realizamos foi a importação de `notasService` e seu uso através de um apelido. Não havia grande necessidade de utilizar esse recurso aqui, a não ser pelo fato de encurtamos o nome do artefato importado. Porém, esse recuso é interessante quando diferentes módulos exportam artefatos de mesmo nome e no momento da importação somos obrigados a lançar mão desse artifício para resolver a ambiguidade.

Nosso código começa a ganhar forma, mas há mais uma melhoria que se torna oportuna. Se precisarmos obter o total de todos os itens a partir de um código em outros lugares da nossa aplicação, repetiremos código. Nesse sentido, vamos isolar a lógica de `sumItems` no próprio serviço a partir de um novo método que utilizará o já existente `listAll()`.

```
// app/nota/service.js

import { handleStatus } from '../utils/promise-helpers.js';

const API = `http://localhost:3000/notas`;

const sumItems = code => notas =>
    notas.$flatMap(nota => nota.itens)
    .filter(item => item.codigo == code)
    .reduce((total, item) => total + item.valor, 0)



export const notasService = {

    listAll() {
        return fetch(API)
        .then(handleStatus)
        .catch(err => {
            console.log(err);
            return Promise.reject('Não foi possível obter as notas fiscais');
        });
    },

    // novo método
    sumItems(code) {

        return this.listAll().then(sumItems(code));
    }
};
```

Por fim, nosso `app.js` ficará assim:

```
import { log } from './utils/promise-helpers.js';
import './utils/array-helpers.js';
import { notasService as service } from './nota/service.js';

document
.querySelector('#myButton')
.onclick = () =>
    service
    .sumItems('2143')
    .then(console.log)
    .catch(console.log);
```

Excelente! Terminamos esse capítulo com um código mais bem organizado. Mais será que podemos torná-lo ainda melhor? É isso que veremos no próximo capítulo.