10

Compose vs Pipe

Transcrição

Na seção anterior criamos a função `compose` para nos ajudar na composição de funções. Ela se coaduna com a definição de composição da matemática, no entanto, para nós programadores, a ordem das funções passadas para compose é atípica. Vamos recapitular esse trecho de código:

```
// código extraído do para flexão
const sumItems = compose(
    sumItemsValue,
    partialize(filterItemsByCode, code),
    getItemsFromNotas
);
```

As funções serão aplicadas da direita para a esquerda, no entanto, pode fazer mais sentido lermos as funções da direita para a esquerda:

```
// não terá o resultado esperado!
const sumItems = compose(
    getItemsFromNotas
    partialize(filterItemsByCode, '2143'),
    sumItemsValue,
);
```

Apesar do código anterior não dar o resultado esperado, a ordem dos parâmetros deixa clara a ordem na qual as funções serão aplicadas. Em sistemas operacionais baseados no Unix, há o operador `|` (pipe) que permite passar a saída de uma operação para a outra e assim sucessivamente. Essa abordagem parece ser mais natural para nós desenvolvedores e podemos conseguí-la criando uma nova função utilitária que chamaremos de `pipe`. Ela é idêntica à `compose`, com a diferença de que as funções serão aplicadas da direita para a esquerda através da função `reduce` e não mais `reduceRight`:

```
// app/utils/operators.js

// código anterior omitido

// nova função!
export const pipe = (...fns) => value =>
  fns.reduce((previousValue, fn) =>
      fn(previousValue), value);
```

Agora, vamos alterar o módulo `app/app.js` para fazer uso da função `pipe` no lugar de `compose`:

```
// app/nota/service.js
import { handleStatus } from '../utils/promise-helpers.js';
// importou pipe no lugar de compose
import { partialize, pipe } from '../utils/operators.js';

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

```
const getItemsFromNotas = notas => notas.$flatMap(nota => nota.itens);
const filterItemsByCode = (code, items) => items.filter(item => item.codigo === code);
const sumItemsValue = items => items.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');
        });

    },

    sumItems(code) {
        // usando pipe e alterando a ordem dos parâmetros
        const sumItems = pipe(
            getItemsFromNotas,
            partialize(filterItemsByCode, '2143'),
            sumItemsValue,
        );
        return this.listAll().then(sumItems);
    }
};
```

Fica ao gosto do desenvolvedor qual função utilizar. A função `pipe` será utilizando ao longo do projeto.