06

Implementando delay em Promises

Transcrição

Promises não possuem nativamente um mecanismo de retry motivo pelo qual teremos que implementar este recurso. Entre cada nova tentativa será preciso realizar um pequeno delay e por este motivo vamos iniciar nossa implementação criando um mecanismo de delay que será utilizado entre chamadas de Promises.

A função delay

Antes de implementarmos nossa função que permitirá realizar um delay entre chamadas de Promises, vamos ver como ela será utilizada em nosso código. Realizaremos um delay de cinco segundos:

```
// app/app.js

// código anterior omitido
const action = operations(() =>
    timeoutPromise(200, service.sumItems('2143'))
    .then(delay(5000)) // chamou delay
    .then(console.log)
    .catch(console.log)
);

// código posterior omitido
```

Na prova de conceito acima, basta encadearmos uma chamada da função delay para que a próxima chamada à `then()` seja postergada. A função recebe como parâmetro o tempo da espera em milissegundos. Todavia, ela deve ser capaz de receber o resultado da chamada anterior e passá-la para o próximo `then()` encadeado. Sem isso, não seremos capazes de obter o total dos itens na chamada `then(console.log)`

Vamos implementar a função `delay` no módulo `app/utils/promise-helpers.js` :

```
// app/utils/promise-helpers.js
// código anterior omitido

export const delay = milliseconds => data =>
    new Promise((resolve, reject) =>
        setTimeout(() => resolve(data), milliseconds)
    );
```

A função `delay` recebe como parâmetro o tempo em milissegundos do delay e possui como retorno outra função. Esta nova função retornada, que lembrará do tempo a ser respeitado, recebe como parâmetro o resultando da chamada à `then()` anterior. Essa chamada pode ou não retornar um valor. Então, a função ao ser invocada retornará uma nova Promise que será resolvida através de uma chamada de `setTimeout` . Quando resolvida, passará o valor recebido da chamada `then()` anterior para sua próxima chamada encadeada.

Vamos alterar `app/app.js` para importar e utilizar nossa função. Nada mais justo do que realizarmos um teste:

```
// app/app.js
// importou delay!
import { log, timeoutPromise, delay} from './utils/promise-helpers.js';
import './utils/array-helpers.js';
import { notasService as service } from './nota/service.js';
import { takeUntil, debounceTime, partialize, pipe } from './utils/operators.js';


const operations = pipe(
    partialize(takeUntil, 3),
    partialize(debounceTime, 500),
);

const action = operations(() =>
    timeoutPromise(200, service.sumItems('2143'))
    .then(delay(5000)) // chamou delay
    .then(console.log)
    .catch(console.log)
);

document
.querySelector('#myButton')
.onclick = action;
```

Excelente! O total será exibido no console cinco segundos após termos clicado no botão.

Agora que já temos nossa função `delay` pronta, chegou a hora de implementarmos nosso mecanismo de retry.