02

Promise.race

Transcrição

Apesar da especificação `Promise` não suportar um mecanismo de `timeout`, podemos implementá-lo com auxílio da função **Promise.race**. Para que possamos ver esse recurso em ação, vamos realizar um teste em `app/app.js`, logo abaixo das importações de módulos:

```
// app/app.js
// importações de módulos omitidas

const promise1 = new Promise((resolve, reject) =>
    setTimeout(() => resolve('promise 1 resolved'), 3000));

const promise2 = new Promise((resolve, reject) =>
    setTimeout(() => resolve('promise 2 resolved'), 1000));

// código posterior omitido
```

Temos duas `Promises` declaradas. A primeira, só será resolvida depois de três segundos, já a segunda depois de um segundo. Todavia, queremos realizar uma "corrida" (race) entre as promises. Nessa corrida, só queremos o resultado daquela que for resolvida primeiro. É aí que entra `Promise.race`.

A função `Promise.race` recebe uma lista de promises e assim que uma delas for resolvida, receberemos imediatamente seu resultado na próxima chamada encadeada à `then`. As demais promises são ignoradas:

```
// app/app.js
// importações de módulos omitidas

const promise1 = new Promise((resolve, reject) =>
    setTimeout(() => resolve('promise 1 resolvida'), 3000));

const promise2 = new Promise((resolve, reject) =>
    setTimeout(() => resolve('promise 2 resolvida'), 1000));

// exibirá no console "promise 2 resolvida";
Promise.race([
    promise1,
    promise2
])
.then(console.log)
.catch(console.log);
```

É importante estar atento que qualquer rejeição que aconteça durante a resolução das Promises direcionará o fluxo da aplicação para dentro da função `catch`. Em suma, estamos interessados no resultado da primeira promise resolvida, mas se algum erro acontecer antes de qualquer resultado válido, caímos dentro do `catch`.

Agora que já entendemos a mecânica de `Promise.race`, podemos remover o teste que fizemos e partir para a implementação do nosso mecanismo de `timeout`.