Aula 18: Promises I

Nessa aula iremos aprofundar nosso conhecimento sobre **operações assíncronas**, ou seja, sobre execução de código de forma assíncrona.

Um forma de ver uma operação assíncrona é:

- Existe um ponto de partida, mas não se sabe **quando** uma resposta será obtida;
- Também não se sabe se a resposta da operação será de **sucesso** ou de **falha**;
- Se o resultado da operação for de **sucesso**, a **resposta esperada** estará disponível para uso pelo sistema;
- Caso o resultado da operação seja de **falha**, um **motivo de falha** estará disponível para uso pelo sistema.

Esse comportamento é implementado pelo objeto **Promise** do Javascript.

Promise

Por definição, uma **Promise** é um **proxy** (um ponto, ou um alvo, onde deve ser concentrado um determinado valor), onde o **valor esperado**, mas não inicialmente conhecido, de uma **operação assíncrona** será concentrado.

O "**valor esperado**" mencionado pode ser um **sucesso**, indicando que a operação ocorreu como esperado, ou uma **falha**, indicando que um erro ocorreu durante a operação assíncrona.

No caso de **falha**, o "**valor esperado**" é considerado como um **motivo de falha**, ou seja, um **erro**.

Uma Promise pode estar num desses três estados:

1. ***pending***: estado inicial, nem **cumprida** nem **rejeitada**;
2. ***fulfilled***: significa que a operação foi **concluída com sucesso**;
3. ***rejected***: significa que a operação **falhou**.

Uma Promise é considerada **resolvida** se possuir estado **cumprida** ou **rejeitada**.

then e catch

Os métodos **then** e **catch** são utilizados para tratamento das respostas da operação assíncrona.

O método **then** recebe dois parâmetros, **onResolved** e **onRejected**:

```
fetch("https://cep.awesomeapi.com.br/json/05424020") .then( (response) => { return response.json() }, // onResolved (error) => { ... "tratamento do erro" ... } // onRejected ) .then( (json) => { console.log(json) } // onResolved (error) => { ... "tratamento do erro" ... } // onRejected );
```

- O método **onResolved** será executado caso a operação resulte em **sucesso**.
- O método **onRejected** será executado caso a operação resulte em **falha**.

É mais comum que se utilizae o método **then** sem o **onRejected**.

```
fetch("https://cep.awesomeapi.com.br/json/05424020") .then( (response) => { return response.json() } // onResolved ) .then( (json) => { console.log(json) } // onResolved );
```

Para o tratamento dos erros que podem ocorrer na execução assíncrona, utiliza-se o método **catch**, que recebe apenas um parâmetro, **onRejected**.

```
fetch("https://cep.awesomeapi.com.br/json/05424020") .then( (response) =>
{ return response.json() } // onResolved ) .then( (json) => {
console.log(json) } // onResolved ) .catch( (error) => { ... "tratamento
do erro" ... } // onRejected );
```

Manipulação do pipeline

É importante observar que, para que um método **then** seja chamado, uma Promise deve ser retornada pelo método anterior.

- No exemplo acima, o método **response.json()** retorna uma nova Promise;
- Essa Promise é retornada pela declaração **return**;
- Dessa forma, o segundo then **captura** essa Promise e realiza os procemintos esperados;
- Se o **return** for removido, o segundo **then** não irá **capturar** a execução da promise retornada pelo método **response.json()**.

É possível manipular o pipeline de execução com os métodos **Promise.resolve(<value>)** e **Promise.reject(<error>)**. Cada método resultará, respectivamente, numa Promise **cumprida** ou numa Promise **rejeitada**.

```
fetch("https://cep.awesomeapi.com.br/json/05424020") .then( (response) =>
{ if (response.ok) { return response.json(); }; return new
Promise.reject("Erro na requisição dos dados!"); } ) .then( (json) => {
return new Promise.resolve(json); } ) .then( (json) => { return new
Promise.resolve(json); } ) .then( (json) => { return new
Promise.resolve(json); } ) .then( (json) => { return new
Promise.resolve(json); } ) .then( (json) => { return new
Promise.resolve(json); } ) .then( (data) => { console.log(data) } )
.catch( (error) => { ... "tratamento do erro" ... } );
```

O objeto Promise

Até o momento, utilizamos o método **fetch** para exemplificar o comportamento de uma **Promise** em Javascript.

Porém, o método **fetch** não é o único a retornar uma **Promise**. Existem diversas situações que necessitam do comportamento caracterizado por **Promises**. E para isso, as vezes se torna necessário **construírmos nossas próprias Promises**.

Para **criar novas Promises**, de acordo com o comportamento que necessitamos, podemos utilizar o objeto **Promise** do Javascript. Tal objeto **retorna uma Promise**, sendo possível então utilizar os métodos **then** e **catch** para gerenciar o **pipeline** de execução da mesma.

```
const value = 2; const minhaPromise = new Promise((resolve, reject) => {
if (x > 5) { resolve({ status: "OK" }); }; reject({ err: "ERROR" }); });
minhaPromise .then(result => { console.log(result); }) .catch(err => { co
nsole.log(err); }); // outuput expected: // { err: "ERROR" }
```

Abstração

Por definição, abstração significa:

```
Ação de abstrair, de analisar isoladamente um aspecto, contido num todo,
sem ter em consideração sua relação com a realidade.
```

O conceito de abstração é muito utilizado na programação. E resumo, significa **levar em consideração apenas o essencial, descartando tudo o que não for importante para o objetivo a ser atingido**.

Podemos fazer uma comparação com o retorno de uma função ou método, que podem analizar uma quantidade enorme de informação, mas no final, **somente o que é importante para o sistema deve ser retornado**.