

Dissecando uma promise!

Revisão de Promise

Para ficarmos melhores do que já somos em *promises*, crie o arquivo `dissecando-uma-promise.html` e cole o código abaixo:

```
<!-- dissecando-uma-promise.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Dissecando uma promise</title>
</head>
<body>
  <script>

    let promise = new Promise((resolve, reject) => {
      setTimeout(() => resolve('PROMISE RESOLVIDA'), 5000);
    });

    promise.then(resultado => console.log(resultado));
  </script>
</body>
</html>
```

Agora, abra a página no Chrome e verifique no console do navegador. Depois de 5 segundos será exibida a mensagem `PROMISE RESOLVIDA`. Mas o que aconteceu durante todo esse processo?

Bom, veja que a variável `promise` recebeu uma instância de `Promise`. O construtor de `Promise` recebe uma função como parâmetro. É essa função passada como parâmetro que será chamada internamente pela `Promise`, quando for criada. Como é a própria `Promise` que chama essa função, ela passa sempre dois parâmetros para ela nesta ordem: a função na qual passamos o valor de sucesso e a função que passamos o valor de fracasso.

```
let promise = new Promise((resolve, reject) => {
  // é aqui dentro que definimos o que será passado para `resolve` e o que será passado para `reject`
});
```

Bom, criar uma `Promise` não é suficiente. Se olharmos o fragmento acima, em nenhum momento estamos dizendo o que acontecerá se a promessa for cumprida. Para efeito didático, colocarei um `setTimeout` de 5 segundos dentro do corpo da `Promise`. Só depois de 5 segundos passaremos o resultado da nossa operação para o `resolve`:

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve('PROMISE RESOLVIDA'), 5000);
});
```

Obtendo o retorno da ação

Perfeito, temos a variável `promise`, que guarda uma instância de `Promise`, o resultado futuro de uma ação. Mas em que parte do código pegamos o resultado dessa ação quando concluída?

É por meio do método `then`, da instância de `Promise` que temos acesso ao resultado da ação. O método `then` recebe uma função e nela temos acesso sempre como primeiro parâmetro ao resultado da ação. Internamente em nossa `Promise`, é o valor passado para `resolve` que estará disponível para a função `then`. Sendo assim, em `then`, só depois de 5 segundos teremos acesso ao resultado a ação, que é uma string, mas poderia ser qualquer outro tipo de dado.

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve('PROMISE RESOLVIDA'), 5000);
});

// imprime no console "PROMISE RESOLVIDA"
promise.then(resultado => console.log(resultado));
```

É interessante saber que, como nosso código é assíncrono, não sabemos quando nossa promessa será cumprida (sabemos que são 5 segundos, mas se fosse uma conexão de rede não teríamos tanta certeza assim, certo?).

Quero que vocês façam uma pequena alteração no código:

```
<!-- dissecando-uma-promise.html -->
<script>

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve('PROMISSE RESOLVIDA'), 5000);
  });

  promise.then(resultado => console.log(resultado));
  console.log('FIM'); // novidade aqui!
</script>
```

Como a `promise` é assíncrona e não bloqueia a execução do nosso código, veremos impresso no console as mensagens nesta ordem:

```
FIM
PROMISE CONCLUÍDA
```

Lidando com erros

E se algo sair errado? Onde trataremos o erro? Quando há algum erro dentro do corpo da nossa `Promise`, cabe ao desenvolvedor capturar esse erro e passá-lo para a função `reject`:

```
<script>

  let promise = new Promise((resolve, reject) => {
    console.log(resolve);
    setTimeout(() => reject('HOVE PROBLEMAS'), 5000);
  });

  promise
```

```
.then(resultado => console.log(resultado));  
</script>
```

Depois de 5 segundos, nossa `promise` será rejeitada, indicando que houve algum erro. Mas onde teremos acesso à causa da rejeição? Basta, depois de `then`, encadearmos uma chamada à função `catch`:

```
<script>  
  
let promise = new Promise((resolve, reject) => {  
  console.log(resolve);  
  setTimeout(() => reject('HOUVE PROBLEMAS'), 5000);  
});  
  
promise  
  .then(resultado => console.log(resultado))  
  .catch(erro => console.log(erro)); // exibe no console HOUVE PROBLEMAS  
</script>
```

Mas é claro que queremos que nossa `promise` esteja preparada para resolver ou rejeitar. Para efeito didático, vamos colocar um variável booleana. Se for `true`, resolvemos, se for `false`, rejeitamos. Dessa forma, você pode brincar e simular quando a `promise` é resolvida ou não:

```
<script>  
  
let ok = false;  
let promise = new Promise((resolve, reject) => {  
  
  // como temos mais de uma instrução, precisamos colocar um bloco em nossa arrow function  
  setTimeout(() => {  
    if(ok) {  
      resolve('PROMISE CONCLUÍDA');  
    } else {  
      reject('HOUVE PROBLEMAS');  
    }  
  }, 5000);  
});  
  
promise  
  .then(resultado => console.log(resultado))  
  .catch(erro => console.log(erro));  
</script>
```