JavaScript Avançado II: ES6, orientação a objetos e padrões de projetos

79%

AULA 05 Combatendo Callback Hell com Promises

ATIVIDADES

O problema da vida assínc... 09min

O padrão de projeto Prom... 14min

Pyramid of Doom novame... 12min

Isolando a complexidade e... 11min

05 Simplificando o código

06 Dissecando uma promise!

07 Declarando uma promise

08 Consolidando seus conhec...

09 Simplificando ainda mais ...

10 Desafio: obter todas as ne...

AULAS

05. Combatendo Callback Hel

OUTROS LINKS

Fórum

Trocar Curso

06

Dissecando uma promise!

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

```
omise.html -->

8">
uma promise</title>

new Promise((resolve, reject) => {
() => resolve('PROMISE RESOLVIDA'), 5000);

esultado => console.log(resultado));
```

ne e verifique no console do navegador. Depois de 5 segundos será exibida a mensagem
aconteceu durante todo esse processo?

se recebeu uma instância de `Promise`. O construtor de `Promise` recebe uma função como
ada como parâmetro que será chamada internamente pela `Promise`, quando for criada. Como
a essa função, ela passa sempre dois parâmetros para ela nesta ordem: a função na qual
a função que passamos o valor de fracasso.

```
se((resolve, reject) => {
e definimos o que será passado para `resolve` e o que será passado para `reje
```

é suficiente. Se olharmos o fragmento acima, em nenhum momento estamos dizendo o que
umprida. Para efeito didático, colocarei um `setTimeout` de 5 segundos dentro do corpo da
dos passaremos o resultado da nossa operação para o `resolve`:

```
se((resolve, reject) => {
esolve('PROMISE RESOLVIDA'), 5000);
```

Farley de Souza Rufino
21.5k xp

ação

JavaScript Avançado II: ES6, orientação a objetos e padrões de projetos

79%

AULA 05 Combatendo Callback Hell com Promises

ATIVIDADES

01 O problema da vida assínc... 09min

02 O padrão de projeto Prom... 14min

03 Pyramid of Doom novame... 12min

04 Isolando a complexidade e... 11min

05 Simplificando o código

06 Dissecando uma promise!

07 Declarando uma promise

08 Consolidando seus conhec...

09 Simplificando ainda mais ...

10 Desafio: obter todas as ne...

AULAS

05. Combatendo Callback Hel

OUTROS LINKS

Fórum

Trocar Curso

ise , que guarda uma instância de `Promise` , o resultado futuro de uma ação. Mas em que ultado dessa ação quando concluída?

la instância de `Promise` que temos acesso ao resultado da ação. O método `then` recebe uma mpre como primeiro parâmetro ao resultado da ação. Internamente em nossa `Promise` , é o ue estará disponível para a função `then` . Sendo assim, em `then` , só depois de 5 segundos ação, que é uma string, mas poderia ser qualquer outro tipo de dado.

```
.se((resolve, reject) => {
esolve('PROMISE RESOLVIDA'), 5000);


'PROMISE RESOLVIDA"
 => console.log(resultado));
```

o nosso código é assíncrono, não sabemos quando nossa promessa será cumprida (sabemos sse uma conexão de rede não teríamos tanta certeza assim, certo?).

equena alteração no código:

```
omise.html -->


romise((resolve, reject) => {
=> resolve('PROMISSE RESOLVIDA'), 5000);


:ado => console.log(resultado));
 // novidade aqui!
```

e não bloqueia a execução do nosso código, veremos impresso no console as mensagens nesta

taremos o erro? Quando há algum erro dentro do corpo da nossa `Promise` , cabe ao rro e passá-lo para a função `reject` :

```
romise((resolve, reject) => {
olve);
=> reject('HOUVE PROBLEMAS'), 5000);
```

JavaScript Avançado II: ES6, orientação a objetos e padrões de projetos

79%

AULA 05 Combatendo Callback Hell com Promises

ATIVIDADES

01 O problema da vida assínc... 09min

02 O padrão de projeto Prom... 14min

03 Pyramid of Doom novame... 12min

04 Isolando a complexidade e... 11min

05 Simplificando o código

06 Dissecando uma promise!

07 Declarando uma promise

08 Consolidando seus conhec...

09 Simplificando ainda mais ...

10 Desafio: obter todas as ne...

AULAS

05. Combatendo Callback Hel

OUTROS LINKS

Fórum

Trocar Curso

```
) => console.log(resultado));
```

`promise` será rejeitada, indicando que houve algum erro. Mas onde teremos acesso à causa da n , encadearmos uma chamada à função `catch` :

```
'romise((resolve, reject) => {
;olve);
=> reject('HOUVE PROBLEMAS'), 5000);
```

```
) => console.log(resultado))
 console.log(erro)); // exibe no console HOUVE PROBLEMAS
```

nossa `promise` esteja preparada para resolver ou rejeitar. Para efeito didático, vamos colocar `true` , resolvemos, se for `false` , rejeitamos. Dessa forma, você pode brincar e simular quando

```
'romise((resolve, reject) => {

iais de uma instrução, precisamos colocar um bloco em nossa arrow function! Le
=> {

e('PROMISE CONCLUÍDA');

'HOUVE PROBLEMAS');
```

```
) => console.log(resultado))
 console.log(erro));
```