

FUNÇÕES ASSÍNCRONAS



DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Sincronia do javascript
- assíncrona e síncrona
- Funções assíncronas (callback)
- Promisses
- async e await

SINCRONIA NO JAVASCRIPT

O javascript é “**single thread**” isso significa que ela executa **uma tarefa de cada vez**, seguindo um **mesmo fio de execução**.

Single thread → único fio, única linha

```
function one() {  
  console.log('A');  
  two();  
  console.log('C');  
}  
function two() {  
  console.log('B');  
}  
one();
```

Call Stack

Console



JAVASCRIPT ASSÍNCRONO

Imaginem a seguinte situação abaixo. Será que é interessante o JavaScript executar de forma sequencial cada uma das instruções?

```
inicializaSistema(); // instantâneo  
carregarUsuario(); // 5 segundos  
carregarConfiguracoes(); // 20 segundos  
montarHomepage(); // instantâneo
```

FUNÇÕES ASSÍNCRONAS

Já no caso de ações assíncronas ele não espera exatamente que um método ou função termine para iniciar o próximo bloco de código, ou seja, nestes casos o multithread (várias ações simultaneamente) é utilizado.

O browser (navegador) disponibiliza uma coleção de funções, e dentre elas, algumas trabalham com chamadas assíncronas. **setTimeout** e **setInterval** são dois exemplos.



PROMISES

Pensando em formalizar um padrão de desenvolvimento para criar operações assíncronas, uma das implementações do ES6 feitas para o javascript foi a criação das chamadas Promises. Antes disso as implementações eram feitas de callbacks().

Declaração:

```
new Promise( (resolve, reject) => { ... } )
```

Retorno caso de
sucesso

Retorno caso de erro



PROMISE

- **resolve()**

Quando a operação da **Promise** é finalizada com **sucesso**, chamamos a função **resolve** com o valor desejado:

```
new Promise((resolve, reject) => {  
  let deuCerto = false;  
  if (deuCerto) {  
    resolve("A promise deu certo");  
  } else {  
    reject("A promise deu errado! ");  
  }  
});
```

PROMISE

- **reject ()**

Quando a operação da **Promise** é finalizada com **erro**, chamamos a função **reject** com o valor desejado:

```
new Promise((resolve, reject) => {  
  let deuCerto = false;  
  if (deuCerto) {  
    resolve("A promise deu certo");  
  } else {  
    reject("A promise deu errado!");  
  }  
});
```


PROMISE

A promise assumir 3 estados diferentes:

- *pending* (pendente) - estado inicial
- *fulfilled* (realizada) - a operação foi concluída com sucesso
- *rejected* (rejeitada) - a operação foi concluída com sucesso, porem foi rejeitada

PROMISE - Exercício

- Pensando em validar dados vindo de um input, crie uma função que irá receber um dado de um input html do tipo text. Caso esse dado seja um número deve ser retornado no método reject da promise a mensagem `"Error no nome, favor inserir algo válido"`, caso não, retorne a mensagem ``Olá ${name} , seja bem vindo`` no método resolve.

A mensagem retornada poderá ser apresentada ou em um console ou na própria tela.

INTERVALO DE AULA

DEV!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:50

Retorno: 21:10



PROMISE (then e catch)

Para capturarem o valor resultante da promise, usamos um método da própria promise chamado `then`.

No uso diário, usamos o **`then`** para capturar apenas as promises que são resolvidas com **`sucesso`**.

Para as promises resolvidas com **`erro`**, usamos outro método, o **`catch`**, que pode ser encadeado junto a um método **`then`**.

PROMISE (then e catch)

Exemplo da sintaxe:

```
minhaPromise.then((valor) => {...}).catch((erro) => {...})
```

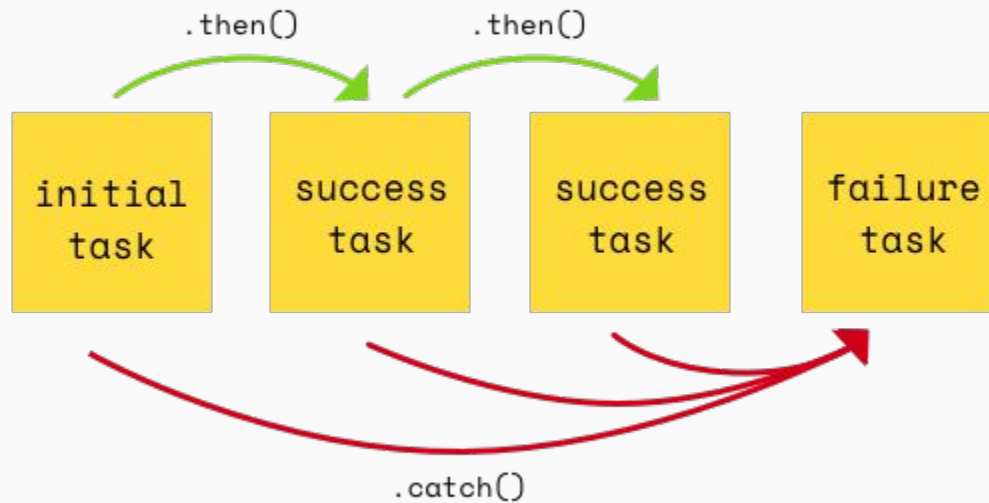
OUU...

Esta função é chamada com o valor resultante da promise em caso de rejected (resolvida com **erro**)

```
minhaPromise.then( (valor) => {...}, (erro) => {...} )
```

Esta função é chamada com o valor resultante da promise em caso de fulfilled (resolvida com **sucesso**)

PROMISE (then e catch)



Exercício

Faça o teste com o `setTimeout` e com o `setImmediate` e observe o que o console irá imprimir.

```
const p = new Promise((resolve, reject) => {
  let valor = 30;
  if (valor > 20) {
    //setTimeout(() => {
      resolve(mostraOi());
    //}, 5000);
  } else {
    reject("Opa deu erro!");
  }
})
console.log(p)
function mostraOi() {
  console.log("amigo estou aquiiaaaaaaaaa");
}
p.then((resolve)=>{
  console.log("amigo estou aqui");
}).catch(
  (err)=>{console.error(err)}
);
```

No ES8 (EcmaScript 2017), o JavaScript introduziu as diretivas `async` e `await` para trabalharmos com `Promise`.


Funções assíncronas (`async functions`) e a diretiva `await` são simplificações de `promises` (conhecida como açúcar sintático, `syntactic sugar`) - tornam o código mais simples e fácil de ler e escrever.

ASYNCE AWAIT

Este código com o método **then** ...

```
const minhaPromise = () => {  
  return Promise.resolve("Resolve promise");  
};
```

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



O método estático "**resolve**" serve para resolver automaticamente um valor (simular uma promise resolvida)

ASYNCR E AWAIT

... é equivalente à esse código com **async** / **await**


```
async function minhaFuncaoAsync() {
```

```
  const resultado = await minhaPromise();
```

```
  console.log("Async:", resultado);
```

```
}
```

```
minhaFuncaoAsync();
```



A execução do código ESPERA nesta linha e aguarda a promise ser **resolvida** com sucesso ou **rejeitada**

ASYNC E AWAIT

Importante: quando iniciamos a declaração de uma função utilizando a palavra-chave **async**, transformamos o **retorno** desta função em **uma promise**.

```
> async function umaPromise ( ) {  
    return "Olá"  
}  
◀ undefined  
  
> umaPromise()  
◀ ▶ Promise {<fulfilled>: 'Olá'}  
  
> umaPromise().then(console.log)  
Olá
```

ASYNC E AWAIT

Por ser uma promise, uma função declarada com `async` também pode ser encadeada com os métodos **.then** e **.catch**:

```
async function dobraValorAsync(value) {  
  return value * 2;  
}
```

```
dobraValorAsync(2)  
  .then(dobraValorAsync())  
  .then(dobraValorAsync())  
  .then(console.log());
```

MATERIAL COMPLEMENTAR

- Programação Assíncrona - [Conceitos de programação Assíncrona](#)
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/async_function
- <https://medium.com/trainingcenter/entendendo-promises-de-uma-vez-por-todas-32442ec725c2>
- https://www.youtube.com/watch?v=7Bs4-rqbCQc&ab_channel=DevPleno

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

[Clique aqui](#) ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.





DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!



<LAB365>