Promises

DigitalHouse>

Índice

“

As **promises** são funções que permitem que o código **assíncrono** seja executado eficientemente.

”

1 Requisição assíncrona

Requisição assíncrona

Uma solicitação assíncrona é um conjunto de instruções executadas por um mecanismo específico, como **callback**, **promise** ou um **evento**. Isto possibilita que a resposta seja processada em outro momento.

Como se pode inferir, seu comportamento é **não-bloqueante,** pois o pedido é executado em paralelo com o resto do código.

2 | .then()

.then()

A função assíncrona retornará um resultado, ou não. Enquanto isso, o código **continua a ser executado**.

{}

```
getUsers()

    .then(function(data){
      console.log(data);


console.log("Ainda funcionando!")
```

Função assíncrona.

Código que poderia continuar a ser executado enquanto a promessa está sendo cumprida.

.then()

Quando a promise é cumprida e um resultado é obtido, o primeiro **.then() é executado** e age somente como uma consequência da função assíncrona.

```
{}
```

```
getUsers()

      .then(function(data){
        console.log(data);
      });

console. log("Ainda funcionando!")
```

Execute o console.log() SOMENTE SE **getUsers()** retorna um resultado. Isto é recebido por **.then()** dentro de seu callback, neste caso, no parâmetro de **dados**.

3 | Cadeia de Promises

Cadeia de Promises

Às vezes, `()` têm muitas promises dentro delas. Para resolver isto, precisamos usar outro `.then()` que entre em execução uma vez que o anterior seja resolvido.

{}

```
getUsers()
        .then(function(data){
         return filterData(data);
    })
        .then(function(dataFiltered){
        console.log(dataFiltered);
    })
```

Cuidado!

É importante lembrar que o **.then()** precisa devolver os dados processados para que possam ser utilizados por outro **.then()**.

.catch()

No caso de **NÃO** se obter um resultado, é gerado um erro. Para isso, utilizamos `.catch()`, que captura quaisquer erros que possam ser gerados através das promises. Dentro deste método, nós decidimos o que fazer com o erro. O erro é recebido como um parâmetro dentro da callback de `.catch()`.

No exemplo a seguir, mostraremos o erro no console:

{}

```
getUsers()
      .then(function(data){
       console.log(data);
      })
      .catch(function(error){
       console.log(error);
    })
```

4 | Promise.all()

Promise.all( )

Às vezes, precisamos de duas ou mais promises para realizar uma determinada ação. Para isso, usamos o `Promise.all()`. Isto conterá uma série de promises que, uma vez resolvidas, executarão um `.then()` com os resultados dessas promises.

A primeira coisa que precisamos fazer é armazenar em variáveis as promises que precisamos obter.

```
{}
```

```
let promiseMovies = getMovies();
```

Promessa de filmes

```
let promiseGenres = getGenres();
```

Promessa de gênero

Promise.all()

O próximo passo é usar o método `Promise.all()` que conterá uma série das promises que salvamos anteriormente.

```
Promessa.all([promiseMovies, promiseGenres])
```

promises a serem resolvidas

{}

Promise.all()

O callback de `.then()` recebe um conjunto de resultados das promises cumpridas.

```
Promessa.all([promiseMovies, promiseGenres])

{}

.then(function([resultFilms,resultGenres]){
  console.log(resultFilms, resultGenres);
})
```

O **.then()** só será executado se ambas as promises forem cumpridas.

Documentação

Para saber mais sobre promises e Promise.all(), você pode acessar a documentação oficial do Mozilla clicando nos links a seguir:

Uso de promises

Promise.all()

DigitalHouse>