



Promises e Fetch API

JavaScript

Aplicações



Como resolver?

- Usando funções callback:

```
funcaoAssincrona(arg1, callback){  
    // faz request e afins  
    // e no final da execução executamos o callback  
    callback();  
}  
  
function callback() {  
    // operação que quero fazer depois que tiver a resposta da requere  
st  
}
```

O problema...

- Múltiplas operações assíncronas levam a códigos ilegíveis (callback hell):

```
obj.funcaoAssincrona(function(response) {  
  response.funcaoAssincrona(function(response2) {  
    response2.funcaoAssincrona(function(response3) {  
      response3.funcaoAssincrona(function(response4) {  
        return response4;  
      });  
    });  
  });  
});
```

Promise

- “Promessa” é um objeto usado para processamento assíncrono
- Representa um valor que pode estar disponível agora, no futuro ou nunca
- Representa um estado para um valor que pode não ser conhecido quando criado.
 - Uma variável criada para receber dados de uma requisição HTTP. Nada garante que a resposta chegará como o esperado...



Estados de uma Promise



PENDING (PENDENTE): ESTADO INICIAL, AINDA NÃO FOI REALIZADA NEM REJEITADA



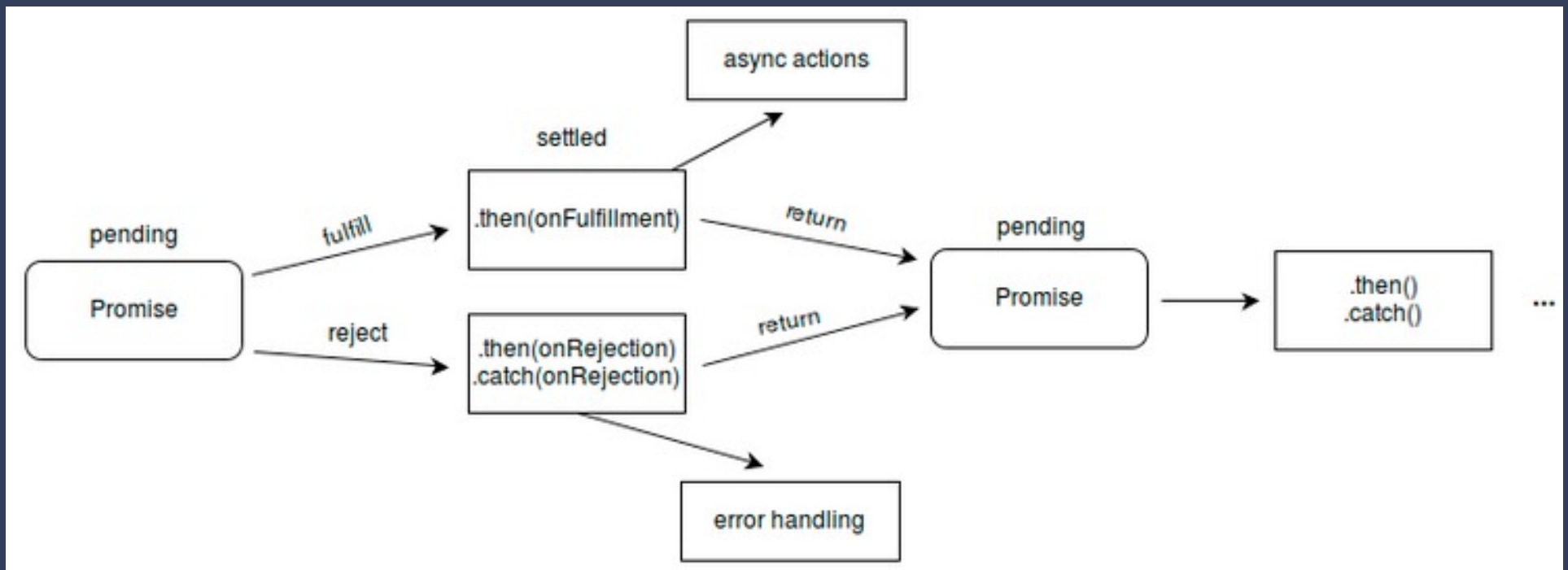
FULFILLED (REALIZADA): SUCESSO NA OPERAÇÃO



REJECTED (REJEITADA): FALHA NA OPERAÇÃO

Fluxo

- Uma promessa pendente pode se tornar *realizada* com um valor ou *rejeitada* por um motivo (erro)
 - Quando um desses estados ocorre (*realizada* ou *rejeitada*), o método **then** é chamado e executa o tratamento associado (**resolved** ou **rejected**)



Métodos

- Promise.all (lista): retorna uma promise que é resolvida quando todas as promises no argumento "lista" forem resolvidas
 - Agrega resultados de múltiplas promises
- Promise.race(lista): retorna uma promise que resolve ou rejeita assim que uma das promises do argumento "lista" resolve ou rejeita
- Promise.reject(motivo): retorna um objeto Promise que foi rejeitado por um dado motivo



Usando promises

- Uma Promise é um objeto que representa a eventual conclusão ou falha de uma operação assíncrona
- É um objeto retornado para o qual você adiciona call-backs, em vez de passar call-backs para uma função

Promises encadeiam tarefas.

Como era, como fica com promises

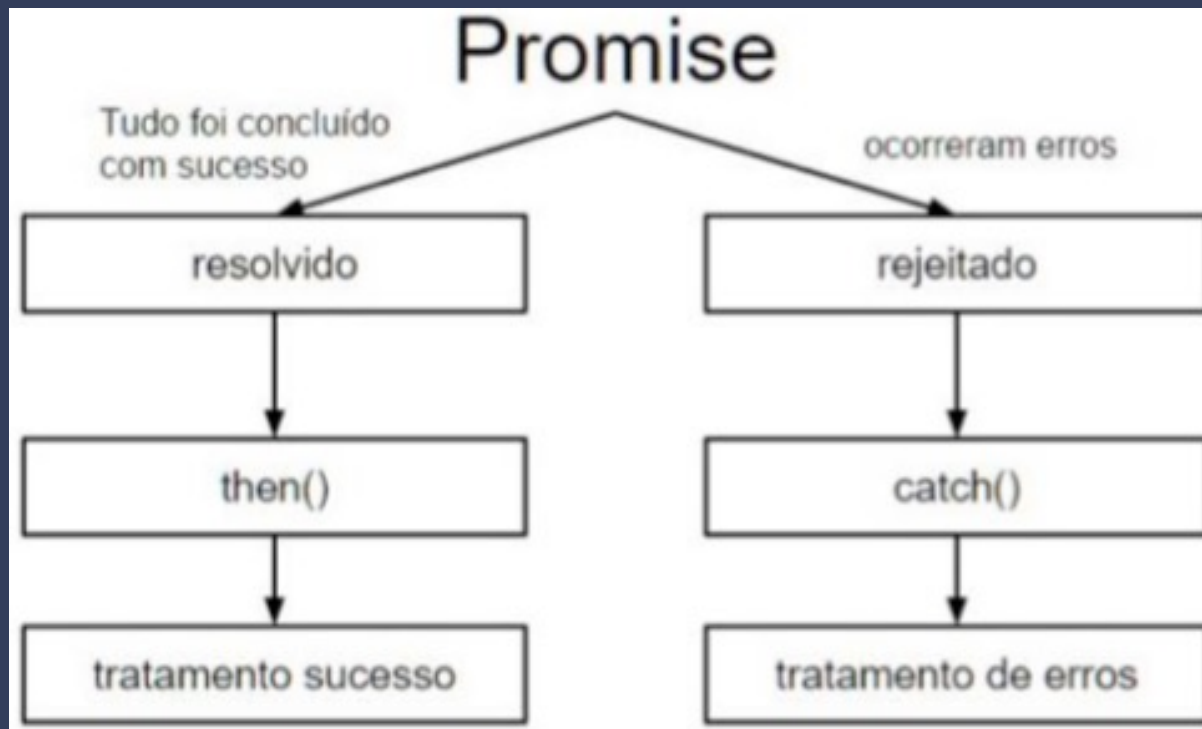
```
function successCallback(result) {  
  console.log("It succeeded with " + result);  
}  
  
function failureCallback(error) {  
  console.log("It failed with " + error);  
}  
  
doSomething(successCallback, failureCallback);
```

```
const promise = doSomething();  
  
promise.then(successCallback, failureCallback);
```

Ou só:

```
doSomething().then(successCallback,  
failureCallback);
```

Fluxo de execução de Promises



Garantias

- Callbacks nunca serão chamados antes da conclusão da execução atual do loop de eventos do JavaScript.
- Callbacks adicionadas com `.then` mesmo *depois* do sucesso ou falha da operação assíncrona, serão chamadas, como acima.
- Múltiplos callbacks podem ser adicionados chamando-se `.then` várias vezes, para serem executados independentemente da ordem de inserção.



Como funciona?

```
let promise = new Promise((resolve, reject) => {  
    // corpo da promise  
});  
  
promise.then();  
promise.catch();
```

Argumentos resolve e reject

- Por padrão, o construtor da promise recebe dois argumentos resolve e reject, para usarmos na lógica quando ela for resolvida ou rejeitada:

```
let promise = new Promise((resolve, reject) => {
  let resultado = true;
  if (resultado) {
    resolve("deu tudo certo!");
  } else {
    reject("deu tudo errado!");
  }
});

promise.then((data) => console.log(`resultado positivo: ${data}`));
promise.catch((data) => console.log(`resultado positivo: ${data}`));
```

Operações assíncronas

```
let promise = new Promise((resolve, reject) => {
  let resultado = false;
  let tempo = 2000; // milissegundos
  setTimeout(() => {
    if (resultado) {
      resolve("deu tudo certo!");
    } else {
      reject("deu tudo errado!");
    }
  }, tempo);
});

promise.then((data) => console.log(`resultado positivo: ${data}`));
promise.catch((data) => console.log(`resultado negativo: ${data}`));

console.log('fui executado antes!');
```

Aninhamento de THEN e CATCH

```
promise
```

```
.then((data) => console.log(`resultado positivo: ${data}`))  
.catch((data) => console.log(`resultado negativo: ${data}`));
```

```
promise
```

```
.then((data) => console.log(`resultado positivo: ${data}`))  
.then((data) => console.log(`resultado positivo 2: ${data}`))  
.catch((data) => console.log(`resultado negativo: ${data}`));
```

```
// errinho no segundo then... o valor de data não é passado para o  
segundo passo
```


Aninhamento de THEN e CATCH

```
promise
  .then((data) => {
    console.log(`resultado positivo: ${data}`)
    return data;
  })
  .then((data) => console.log(`resultado positivo 2: ${data}`))
  .catch((data) => console.log(`resultado negativo: ${data}`));

// agora sim!
```

A interface Fetch

- Fornece uma interface JavaScript para acessar e manipular partes do pipeline HTTP
- Facilita a busca de recursos de forma assíncrona através da rede
- Alternativa moderna ao objeto XMLHttpRequest



Verificando compatibilidade

```
if(self.fetch) {  
    // execute minha solicitação do fetch aqui  
} else {  
    // faça alguma coisa com XMLHttpRequest?  
}
```

Fazendo requisições via Fetch

- Inserindo uma imagem em um elemento

```
var myImage = document.querySelector('img');

fetch('flowers.jpg')
  .then(function(response) {
    return response.blob();
  })
  .then(function(myBlob) {
    var objectURL = URL.createObjectURL(myBlob);
    myImage.src = objectURL;
  });
```

Verificando se o fetch foi bem sucedido

```
fetch('flowers.jpg').then(function(response) {
  if(response.ok) {
    response.blob().then(function(myBlob) {
      var objectURL = URL.createObjectURL(myBlob);
      myImage.src = objectURL;
    });
  } else {
    console.log('Network response was not ok.');
```

Fornecendo seu próprio objeto de solicitação

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request('flowers.jpg', myInit);

fetch(myRequest)
  .then(function(response) {
    return response.blob();
  })
  .then(function(myBlob) {
    var objectURL = URL.createObjectURL(myBlob);
    myImage.src = objectURL;
  });
```