Plataforma Node.js

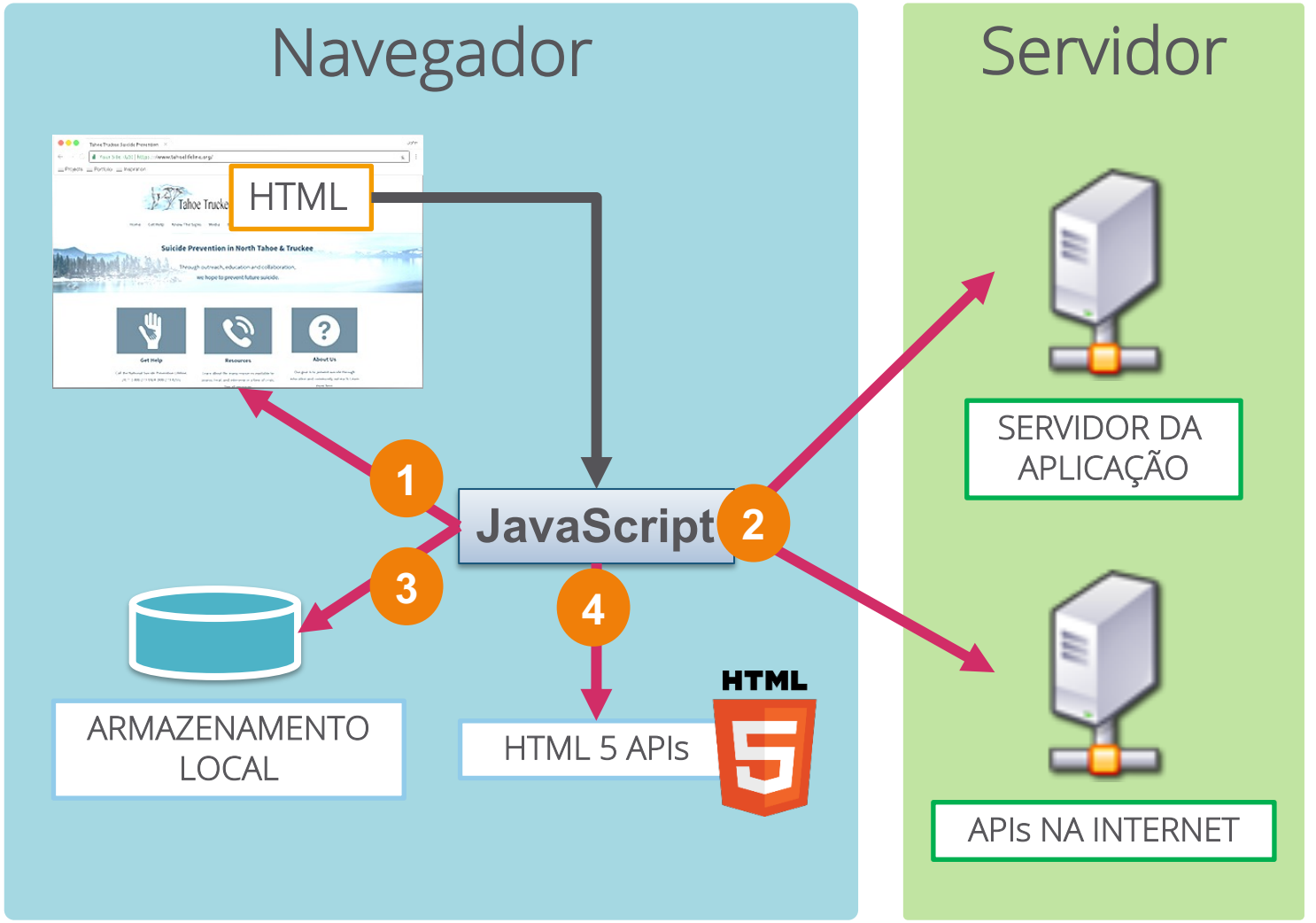
Linguagem JavaScript

Prof. Rommel Vieira Carneiro

Browsers Engines e ECMAScript engines

Browser / Ambiente	Web Engine	ECMAScript Engine
Mozilla Firefox	Gecko	Spider Monkey
Google Chrome	Blink	Google V8
Apple Safari	WebKit	JavaScriptCore
Internet Explorer	Trident	Chakra Core
Edge	EDGE	Chakra Core | Google V8

Capacidades do JavaScript no Navegador



1. Manipulação de objetos da página HTML e tratamento de eventos (**DOM Document Object Model)**
2. Comunicação com o servidor e uso de Application Programming Interface (API) via AJAX (**XMLHttpRequest | Fetch**)
3. Persistência de dados em estruturas providas pelo Browser **(Indexed DB e LocalStorage)**
4. Interação com recursos das novas APIs do HTML 5 (**Canvas, Media, File, Drag and Drop, Geolocation, Web Workers, History**)

Let e Const

- **var** define elementos de escopo global ou escopo de função, se for dentro de funções
- **let** e **const** definem elementos de escopo local.

```
var z = 'original Z';
{
    let x = 'original X';
    var y = 'original Y';
    console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
    {
        // novo escopo ==> novo x
        const x = "novo---- X";
        console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
    }
    // retorno ao escopo anterior ==> original x
    console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
}
console.log(`z: ${z}`); // z é global
console.log(`y: ${y}`); // y é global
console.log(`x: ${x}`); // x é local e linha gera erro
```

Template Literals

- Similar a strings, com crases (`` `..` ``) no lugar de apostrofes (`'..'`) ou aspas (`".."`)
- Permite textos em múltiplas linhas
- Permite a inserção de expressões por meio da construção `${ expr }`

```
// Basic literal string creation
let s1 = `In JavaScript '\n' is a line-feed.`

// Multiline strings
let s2 = `In JavaScript this is
not legal.`

// String interpolation
var name = "Bob", time = "today";
let s3 = `Hello ${name}, how are you ${time}?`

console.log(`${s1} \n${s2} \n${s3}`);
```

Arrow Functions

Arrow Functions são uma sintaxe simplificada para a definição de funções.

```
// forma tradicional de declaração
soma = function (a, b) { return a + b }

// Declaração com arrow functions
soma = (a, b) => { return a + b }

// ou sem as chaves
soma = (a, b) => a + b
```

Arrow Functions

É importante ressaltar que o operador **`this`** em uma Arrow Function faz referência ao bloco que contém a função, diferentemente de funções normais.

```
// Lexical this
var bob = {
    _name: "Bob",
    _friends: [],

    printFriends() {
        this._friends.forEach(f =>
            console.log(this._name + " knows " + f));
    }
}
```

Arrow Functions – Exemplos

```
// Expression bodies
evens = [0, 2, 4, 6, 8, 10, 12, 14, 16];
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({ even: v, odd: v + 1 }));

// Statement bodies
fives = [];
nums.forEach(v => {
    if (v % 5 === 0)
        fives.push(v);
});
```

Parâmetros default

Funções agora suportam definição de valores padrão para parâmetros não enviados

```
function foo(index = 0, testing = true) {
    console.log(`index: ${index} | testing: ${testing} `);
};

foo(); // Imprime ==> index: 0 | testing: true
foo(5); // Imprime ==> index: 5 | testing: true
foo(8, false); // Imprime ==> index: 8 | testing: false
```

Destructuring

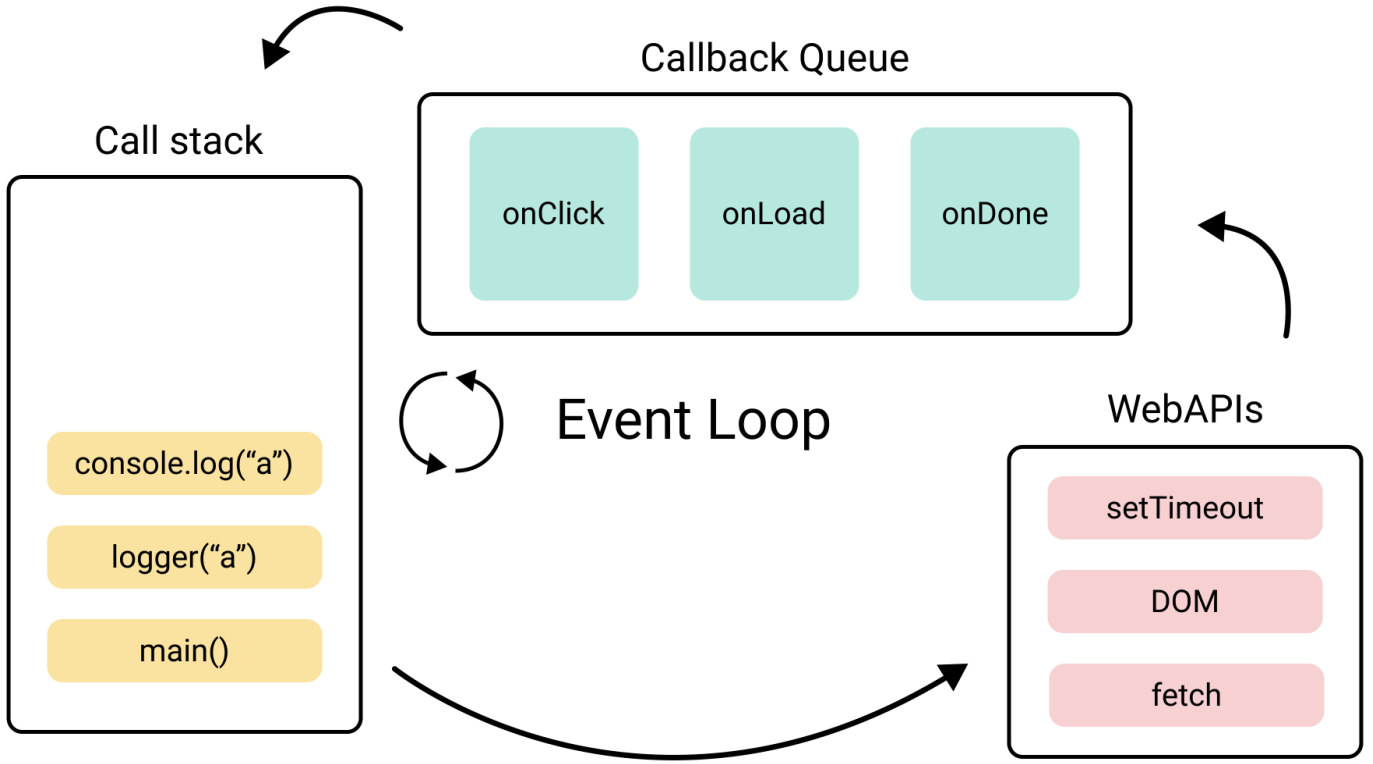
Permite a associação de elementos utilizando casamento de padrões

```
// list matching
var [a, b, c] = [1, 'orange', true];
console.log(`a: ${a} | b: ${b} | c: ${c}`)
// Imprime ==> a: 1 | b: orange | c: true

// Can be used in parameter position
function g({ name }) {
    console.log(`name: ${name}`); // Imprime ==> name: 5
}
g({ name: 5 })

// Fail-soft destructuring
var [a] = [];
console.log(`a: ${a}`); // Imprime ==> a: undefined

// Fail-soft destructuring with defaults
var [a, b = 1] = [2];
console.log(`a: ${a} | b: ${b}`); // Imprime ==> a: 2 | b: 1
```

Arquitetura JavaScript

Componentes

- Call Stack
- Event Loop
- Calback Queue
- Web APIs



Fontes:

- Javascript Event Loop Explained
- Mas que diabos é o loop de eventos?
- Loupe
- JavaScript Concurrency Model and Event Loop
- Modelo de Concorrência e Event Loop

Arquitetura – Single Thread JavaScript

Desafio → Qual é a ordem de saída do texto no console e em que tempo?

```
setTimeout(() => console.log('A'), 0);
console.log('B');
setTimeout(() => console.log('C'), 100);
setTimeout(() => console.log('D'), 0);

let i = 0;
while (i < 1000000000) { // Assume this takes ~500ms
  let ignore = Math.sqrt(i);
  i++;
}

console.log('E');
```

Log	B	E	A	D	C
Time	1ms	501ms	502ms	502ms	502ms

Fonte: [Distributed Systems with Node.js](#) - Thomas Hunter (2020)

Promises – Programação Assíncrona

Modelo Assíncrono Tradicional

```
function successCallback(result) {
  console.log("It succeeded with " + result);
}

function failureCallback(error) {
  console.log("It failed with " + error);
}


doSomething(successCallback, failureCallback);
```

Modelo Assíncrono com Promises

```
function successCallback(result) {
  console.log("It succeeded with " + result);
}

function failureCallback(error) {
  console.log("It failed with " + error);
}

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

Fontes

- Usando promises - JavaScript | MDN (mozilla.org)

Promises – Programação Assíncrona

Modelo Assíncrono Tradicional

```
doSomething(function(result) {
    doSomethingElse(result, function(newResult) {
      doThirdThing(newResult, function(finalResult) {
        console.log('Final: ' + finalResult);
      }, failureCallback);
    }, failureCallback);
  }, failureCallback);
```

Modelo Assíncrono com Promises

```
doSomething()
.then(result => doSomethingElse(result))
.then(newResult => doThirdThing(newResult))
.then(finalResult => {
   console.log(`Final: ${finalResult}`);
 })
.catch(failureCallback);
```

Fontes

- Usando promises - JavaScript | MDN (mozilla.org)

Promises

Uma **Promise** é um mecanismo que recebe uma função que por sua vez apresenta dois callbacks:

1. Uma operação **resolve** a ser executada caso a função seja bem sucedida
2. Uma operação **reject** a ser executada caso a função falhe

As Promises buscam simplificar a **programação assíncrona**. Recursos similares realizados, anteriormente, via bibliotecas como: jQuery ou deferred.js.

```
let p = new Promise(function (resolve, reject) {
    // Executa operação demorada

    if (/* operação bem sucedida  */) {
        resolve("Valor a ser procesado");
    }
    else { /* operação falhou */
        reject(Error("A operação falhou"));
    }
});
```

Fontes

- Promises in JavaScript explained whimsically - https://medium.com/@kevinyckim33/what-are-promises-in-javascript-f1a5fc5b34bf
- Working With Promises (Google) - https://developers.google.com/web/ilt/pwa/working-with-promises
- Promises Explained - http://www.thedevnotebook.com/2017/02/javascript-promises.html

Promises – Exemplo

```
function loadImage(url) {
    // Gera uma Promise para carregar uma imagem
    return new Promise(function (resolve, reject) {
        // Cria uma imagem e atribui sua fonte (src)
        var image = new Image();
        image.src = url;

        image.onload = function () {// Se der erro, executa callback resolve (then)
            resolve(image);
        };
        image.onerror = function () {  // Se der erro, executa callback reject (catch)
            reject(new Error('Could not load image at ' + url));
        };
    });
}
```

Função que retorna uma Promise para tratar eventos após o carregamento de uma imagem

Exemplo de Uso

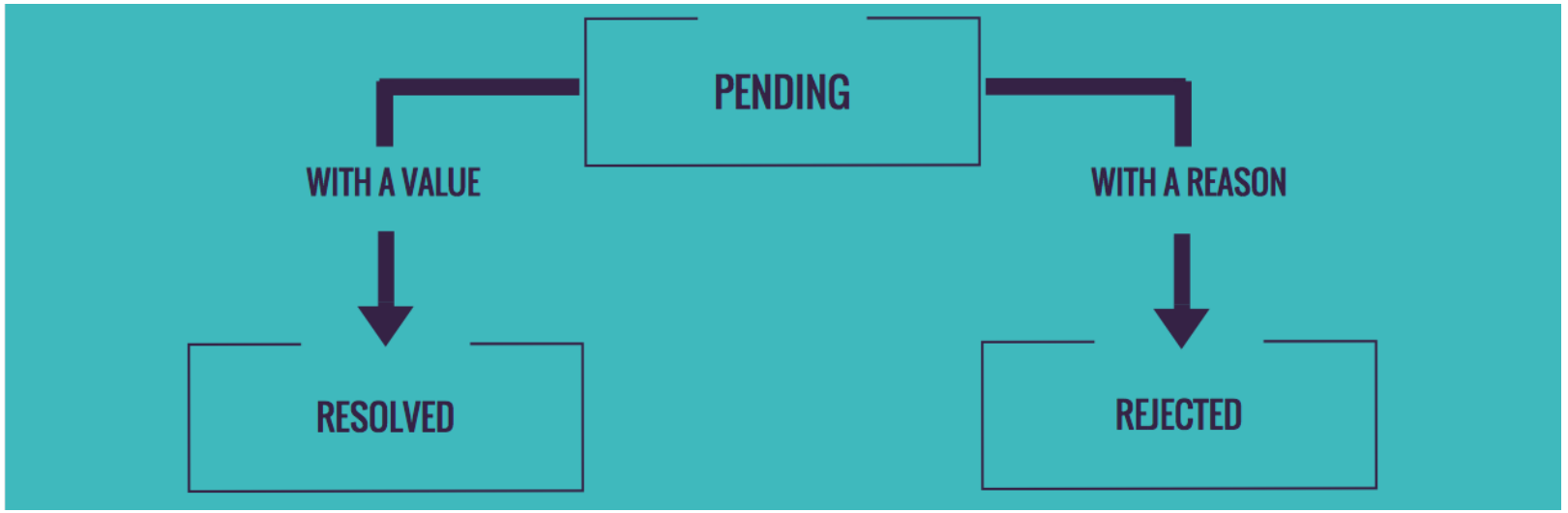
```
loadImage ('http://lorempixel.com/200/200/people')
    .then (imgElem => document.body.appendChild (imgElem))
```

Fonte: Working With Promises (Google) - https://developers.google.com/web/ilt/pwa/working-with-promises

Promises

<u>Estados de uma promise</u> – Possíveis estágios de uma promise no tempo

1. Pendente (pending) – A operação disparada ainda está sendo executada
2. Resolvida (resolved) – A operação teve êxito e executa-se a função **resolve**
3. Rejeitada (rejected) – A operação falhou e executa-se a função **reject**



Promises

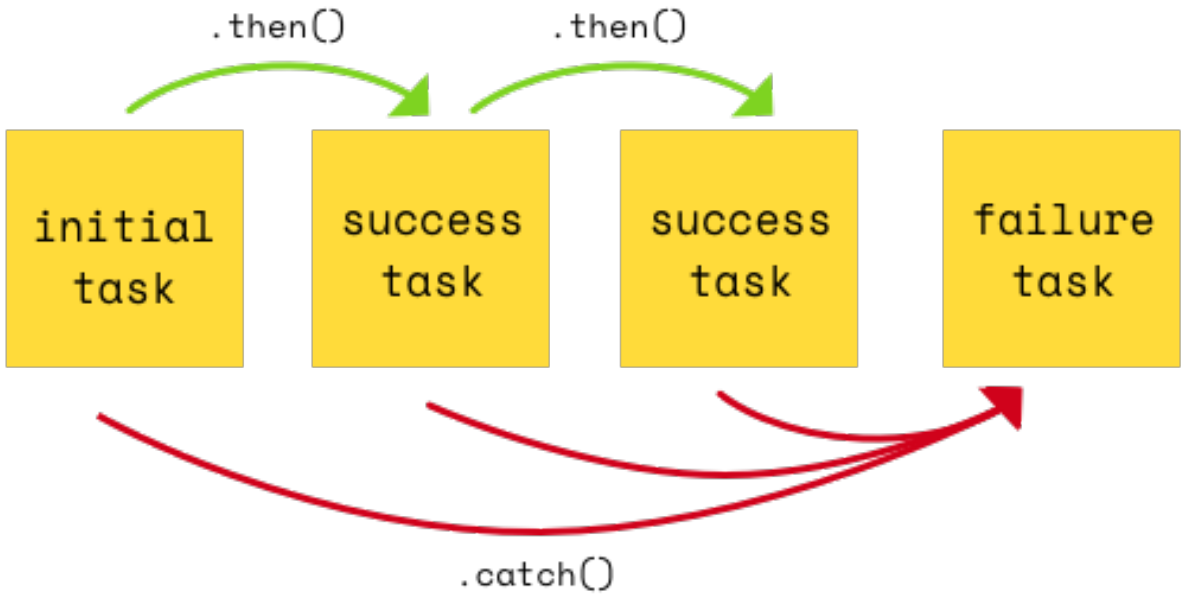
Encadeamento de Promises

Ao receber uma Promise, pode-se disparar

o método **then** para executar uma operação no caso de sucesso ou

o método **catch** no caso de uma falha.

Os dois métodos retornam outra Promise, permitindo o encadeamento de novas operações.



Promises – Exemplo – Fetch API

Encadeamento de Promises – Passo a passo da requisição AJAX com Fetch

1. Fetch executa requisição AJAX, e retorna uma Promise objeto de **resposta (res)**
2. Via **then**, encadeamos função para tratar a **resposta (res)**, retornamos um **JSON (obj)**
3. Via **then**, encadeamos função que trata **obj**, um array, e converte em outro array **usernames**
4. Via **then**, encadeamos função que imprime array **usernames** de nomes no console

```
1   fetch('https://jsonplaceholder.typicode.com/users')
2       .then(res => res.json())
3       .then(obj => obj.map(user => user.username))
4       .then(userNames => console.log(userNames));
```

Promises – All

Promise.all permite criar tratamentos que são executados quando várias promises forem concluídas.

O resultado passado como parâmetro para a função de callback possui um array com o retorno de cada uma das promises processadas.

```
Promise.all ([
    fetch('http://httpbin.org/delay/5'),
    fetch('http://httpbin.org/delay/3'),
]).then(result => console.log('Ok', result))
```

Fonte: [Programação Assíncrona em JavaScript - do básico ao avançado - Speaker Deck]

Referências

- **ECMAScript 2015 Language Specification**
  http://www.ecma-international.org/ecma-262/6.0/
- **The ES6 Guide**
  https://flaviocopes.com/es6/
- **ECMAScript 6 New Features: Overview & Comparison**
  http://es6-features.org
- **Exploring ES6**
  http://exploringjs.com/es6/
- **ECMAScript 6 - Luke Hoban ➔ Fonte de parte dos códigos de exemplo deste material**
  https://github.com/lukehoban/es6features
- **Top 10 ES6 features by example**
  https://blog.pragmatists.com/top-10-es6-features-by-example-80ac878794bb
- **Top 10 ES6 Features Every Busy JavaScript Developer Must Know**
  https://webapplog.com/es6/