

Aula 3 - JS, Node.js e Express

Tópicos da aula

- Javascript
 - conhecendo a linguagem: conceitos básicos
 - ES6+
- Node.js
 - vantagens e uso
 - pq usar javascript no backend?
 - ambiente de desenvolvimento
- Express

Javascript

- Linguagem de programação interpretada, de alto nível e dinâmica
- Lado do cliente e do servidor (**node.js**)
- Dinamicamente tipada
- Baseada em protótipos
- Funcional e Orientada a objetos

Conceitos básicos

- variáveis: var, let e const;
- tipos de dados: number, string, boolean, null, undefined, **array**, **funções**...
- operadores:
 - aritméticos
 - comparação
 - lógicos
- estruturas de controle
 - if, else if, else e switch
 - for, while, do...while
- funções

```
function soma(a, b) {  
  return a + b;  
}
```

```
const saudacao = function (nome) {  
  return `Olá, ${nome}!`;  
};  
  
const multiplicar = (a, b) => a * b;
```

O que o ES6+ trouxe de mais importante?

- Classes
 - sintaxe para criação de objetos e herança baseada em protótipos
- Modules
 - permitiu a exportação e importação de código entre arquivos;

```
1 class Person {  
2   constructor(name, age) {  
3     this.name = name;  
4     this.age = age;  
5   }  
6   ..  
7   message() {  
8     return `My name is ${this.name} and i have ${this.age} old.`;  
9   }  
10 }  
11  
12 const p1 = new Person("Alice", 25);  
13 console.log(p1.message());
```

```
1 export const name = "Alice";  
2  
3 export function message() {  
4   console.log("Hello world!");  
5 }  
6  
7 import { name, message } from './modulo.js';  
8 message();
```

Classes

- Forma de implementar OOP em Javascript;

```
1 class Person {  
2   constructor(name, age) {  
3     this.name = name;  
4     this.age = age;  
5   }  
6   ...  
7   message() {  
8     return `My name is ${this.name} and i have ${this.age} old.`;  
9   }  
10 }  
11  
12 const p1 = new Person("Alice", 25);  
13 console.log(p1.message());
```

- Introduziu os conceitos de herança

```
1 class Student extends Person {  
2   constructor(name, age, course) {  
3     super(name, age);  
4     this.course = course;  
5   }  
6   ...  
7   StudentInfo() {  
8     return `${this.message()}. I'm studying ${this.course}`;  
9   }  
10 }  
11  
12 const Student1 = new Student("Bob", 22, "Prog web");  
13 console.log(Student1.StudentInfo());
```

- Promises:
 - um objeto que representa a eventual conclusão de uma operação assíncrona e seu valor resultante;
 - pendente, cumprida e rejeitada
 - then(onFulfilled, onRejected), catch(onRejected), finally(onFinally)

```
1 const myPromise = new Promise((resolve, reject) => {  
2   let sucesso = true;  
3  
4   if (sucesso) {  
5     resolve("Successfull!");  
6   } else {  
7     reject("Error");  
8   }  
9 });
```

```
11 myPromise  
12   .then((message) => {  
13     console.log(message);  
14   })  
15   .catch((error) => {  
16     console.error(error);  
17   })  
18   .finally(() => {  
19     console.log('Successfull!');  
20   });
```

```

1  const url = "https://api.exemplo.com/data";
2
3  function searchData(url) {
4    return new Promise((resolve, reject) => {
5      fetch(url)
6        .then((response) => {
7          if (!response.ok) {
8            throw new Error("Network error");
9          }
10         return response.json();
11       })
12       .then((data) => resolve(data))
13       .catch((error) => reject(error));
14    });
15  }
16
17  searchData(url)
18    .then((data) => {
19      console.log("response data:", data);
20    })
21    .catch((error) => {
22      console.error("Error to get data:", error);
23    });

```

Promise.all
 Promise.race
 Promise.allSettled
 Promise.any

Qual foi o verdadeiro ganho das Promises em Javascript?

- Código assíncrono era frequentemente gerenciado por meio de callbacks;
- Complexidade do código e a difícil manutenção

```
1  doSomething(function (result) {  
2    doSomethingElse(  
3      result,  
4      function (newResult) {  
5        doThirdThing(  
6          newResult,  
7          function (finalResult) {  
8            console.log(finalResult);  
9          },  
10         failureCallback  
11       );  
12     },  
13     failureCallback  
14   );  
15 }, failureCallback);
```

```
1  doSomething()  
2    .then((result) => doSomethingElse(result))  
3    .then((newResult) => doThirdThing(newResult))  
4    .then((finalResult) => console.log(finalResult))  
5    .catch((error) => console.error(error));
```

- Melhor tratamento de erros
- Encadeamento mais simples;
- Operações paralelas;
- Compatibilidade com **async/await**

Async/await

- Introduzida no ES8;
- Permitiu que código assíncrono fosse escrito de maneira síncrona;

```
1  async function fetchData() {  
2    try {  
3      const response1 = await fetch("url1");  
4      const data1 = await response1.json();  
5  
6      const response2 = await fetch("url2");  
7      const data2 = await response2.json();  
8  
9      console.log("Data:", data1, data2);  
10   } catch (error) {  
11     console.error("Erro to get data info:", error);  
12   }  
13 }  
14  
15 fetchData();
```

- Grandes benefícios ao gerenciamento de operações assíncronas em JavaScript;
- Evitar o callback hell;

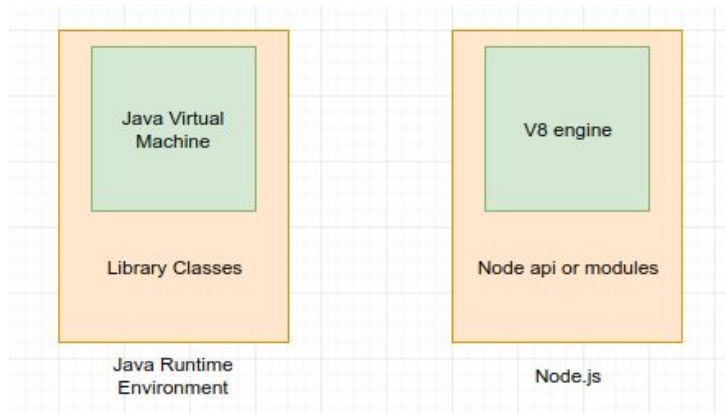
```
1  const fs = require("fs");
2
3  fs.readFile("file1.txt", "utf8", function (err, data1) {
4    if (err) {
5      console.error(err);
6      return;
7    }
8    processFile(data1, function (err, processedData) {
9      if (err) {
10        console.error(err);
11        return;
12      }
13      fs.writeFile("file2.txt", processedData, function (err) {
14        if (err) {
15          console.error(err);
16          return;
17        }
18        console.log("File written successfully");
19      });
20    });
21  });
```

- Legibilidade
- Manutenção
- Tratamento de erros
- Reusabilidade

```
1  const fs = require('fs').promises;
2
3  async function processFiles() {
4    try {
5      const data1 = await fs.readFile('file1.txt', 'utf8');
6      const processedData = await processFile(data1);
7      await fs.writeFile('file2.txt', processedData);
8      console.log('File written successfully');
9    } catch (err) {
10      console.error(err);
11    }
12  }
13
14  async function processFile(data) {
15    return new Promise((resolve, reject) => {
16      const processedData = data.toUpperCase();
17      resolve(processedData);
18    });
19  }
20
21  await processFiles();
```

Node.js

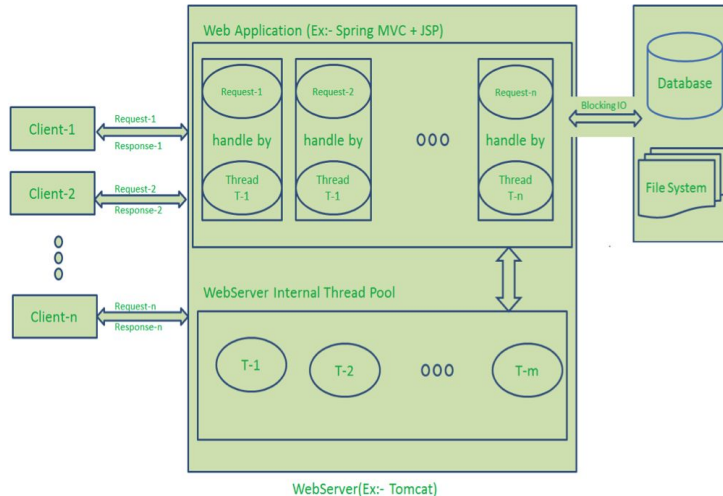
- Ambiente de execução JavaScript assíncrono baseado no V8 do Google Chrome;
- Sair do navegador e executar em máquinas próprias, como uma aplicação autônoma;



- V8 engine transmit código baixo nível para o computador, que não tem a necessidade de interpretá-lo primeiramente;

Node.js

- Single-threaded, Event-driven - Modelo de Loop de Eventos com Thread Única
- Request-Response Multi-threaded;



1. cliente envia requisições para o servidor web (SW);
2. SW mantém um pool de threads limitados;
3. SW em loop infinito esperando;
4. SW recebe a requisição e a atribui a uma thread;

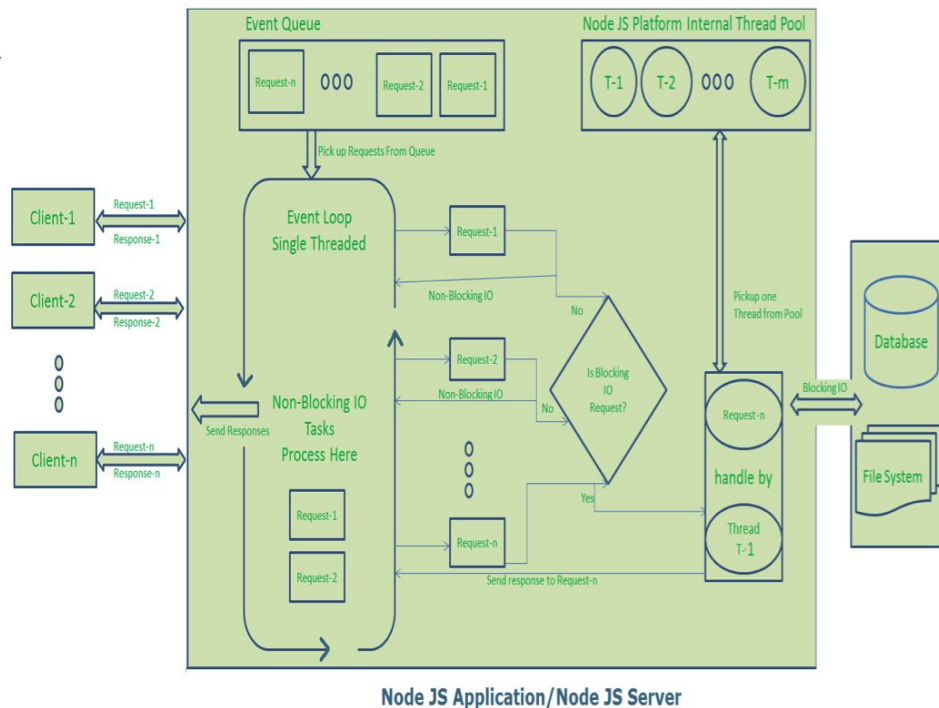
desvantagem: cria uma req por cliente, ocupa as threads.

Node.js

- Single-threaded, Event-driven - Modelo de Loop de Eventos com Thread Única

baseado no modelo de eventos do Javascript;

1. Clientes enviam req para o SW;
2. SW mantém um pool de threads;
3. Event Queue;
4. Event Loop (EL);
 - a. utiliza apenas uma thread;
5. Se tiver req na Queue;
 - a. processa e se não houver IO bloqueante atribui a uma thread;
 - b. a thread processa e retorna ao EL;
6. EL devolve ao cliente;



Node.js - NPM (node package manager)

- Maior ecossistema de bibliotecas de código aberto do mundo;
- Facilita a instalação e gerenciamento de pacotes;

```
{
  "name": "my_project",
  "version": "1.0.0",
  "description": "A little description of my project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified.\" && exit 1"
  },
  "author": "Glauber",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Node.js - vantagens

- Alta performance devido ao V8 e ao modelo non-blocking I/O
- Escalabilidade
- Linguagem server-side e client-side;
- Javascript é uma das linguagens mais utilizadas atualmente;

Express

- Framework web para Node.js e fornece um conjunto robusto de recursos;
- Abstrai funcionalidades como:
 - roteamento e middleware

```
app.get("/", (req, res) => {  
  res.send("Página Inicial");  
});  
  
app.post("/submit", (req, res) => {  
  res.send("Dados enviados");  
});  
  
app.put("/update/:id", (req, res) => {  
  res.send(`Atualizar item com ID: ${req.params.id}`);  
});  
  
app.delete("/delete/:id", (req, res) => {  
  res.send(`Deletar item com ID: ${req.params.id}`);  
});
```

```
1  const express = require('express');  
2  const app = express();  
3  
4  // Application Middleware  
5  app.use((req, res, next) => {  
6    console.log(`${req.method} ${req.url}`);  
7    next();  
8  });  
9  
10 // Route Middleware  
11 app.get('/usuario/:id', (req, res, next) => {  
12   const userId = req.params.id;  
13   if (isNaN(userId)) {  
14     res.status(400).send('User id invalid');  
15   } else {  
16     next();  
17   }  
18 }, (req, res) => {  
19   const userId = req.params.id;  
20   res.send(`Usuário ID: ${userId}`);  
21 });  
22  
23 // Manipulate error Middleware  
24 app.use((err, req, res, next) => {  
25   console.error(err.stack);  
26   res.status(500).send('Something wrong!');  
27 });
```

Express

- Trabalhando com dados:
 - O Express facilita a manipulação de dados em requisições HTTP usando middlewares como body-parser

```
1  const bodyParser = require("body-parser");
2  app.use(bodyParser.json());
3
4  app.post("/users", (req, res) => {
5    const user = req.body;
6    res.send(`User ${user.name} was created!`);
7  });
8
```

Lab 01 -

link:

<https://github.com/joseglauberbo/ProgWeb>

Referências

- FLANAGAN, David. **JavaScript: o guia definitivo**. Bookman Editora, 2012.
- <https://www.digitalocean.com/community/tutorials/node-js-architecture-single-threaded-event-loop>
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Node_server_without_framework
- <https://docs.npmjs.com/>