front-end

prof. Lucas Ferreira

engenharia de software

engenharia de computação

Introdução ao Front-end: JavaScript e AJAX

Java... Script?

- **JavaScript** ou **JS** é uma linguagem de programação interpretada
- Juntamente com HTML e CSS, o JavaScript é uma das três principais tecnologias WEB
- É uma linguagem amplamente usada em navegadores web (client-side)
- E também usada em servidores através de Node.js (exemplo)
- É linguagem multi-paradigma com suporte a estilos de programação orientados a eventos, funcionais e imperativos (orientado a objetos e prototype-based)
- É baseada em ECMAScript *, padronizada pela Ecma international nas especificações ECMA-262[6] e ISO/IEC 16262
- ...e não têm nada haver com Java!

Breve História

- Criada por Brendan Eich na década de 90, mais precisamente em 1995
- Encomendada pela Netscape Communications, Eich escreveu a linguagem em 10 dias, em maio de 1995
- A Netscape acreditava que a web teria que ser mais dinâmica, pois o Navigator tinha sempre que fazer uma requisição ao servidor para obter uma resposta no estado de navegação
- Desenvolvida sob o nome Mocha, a linguagem chegou a ser chamada de LiveScript
- Mas com o lançamento do Netscape Navigator 2.0 em setembro de 1995, mas foi renomeada para JavaScript
- Atualmente encontra-se na versão estável **ES2015 (ES5)** porém já com diversos recursos de versões futuras a disposição.

Características

- O uso primário de JavaScript é escrever funções que são embarcadas ou incluídas em páginas HTML e que interagem com o Modelo de Objeto de Documentos (DOM) da página.
- **Suporte universal:** Todos os navegadores da Web modernos e populares suportam JavaScript com interpretadores integrados.
- **Imperativa e Estruturada:** JavaScript suporta os elementos de sintaxe de programação estruturada da linguagem C como, por exemplo, if, while, switch.

Linguagem "dinâmica"

- **Tipagem dinâmica** 👉 tipos são associados com valores, não com variáveis
- **Baseada em objetos**
- Avaliação em tempo de execução 👉 ***eval***

Linguagem "Funcional"

- **Funções de primeira classe** 👉 objetos que possuem propriedades e métodos
- **Funções aninhadas** 👉 são funções definidas dentro de outras funções

Linguagem "Baseada em Protótipos"

- **Protótipos** 👉 mecanismo de herança semelhante a classes
- **Funções e métodos** 👉 não há distinção entre a definição de uma função e a definição de um método no JavaScript

Mas para que serve o JavaScript?

- Interação com elementos de uma página HTML (DOM)
- Trabalhar com variáveis, resultados e lógica
- Proporcionar interações ricas ao usuário
- Requisitar dados e informações do servidor sem recarregar a página (AJAX)
- Desenvolver aplicativos mobile (IONIC e React Native)
- E porque não também servir páginas e documentos da web (Node.js)?

Hello World

```
function greetMe(nome) {
  alert("Olá, " + nome);
}

greetMe("mundo"); // "Olá, mundo"
```

Sintaxe Básica

Primeiro ponto a ser destacado é que JavaScript é case-sensitive:

```
// abaixo temos duas variáveis distintas
var Nome = "Wesley";
var nome = "Vinícius";
```

Declaração de Variáveis

`var` 👉 Declara uma variável

`let` 👉 Declara uma variável local de escopo do bloco

`const` 👉 Declara uma constante de escopo de bloco, *apenas de leitura*

```
if (true) {
  var x = 5;
}
console.log(x); // 5
```



```
if (true) {
  let y = 5;
}
console.log(y); // ReferenceError: y não está definido
```

Tipos de dados (primitivos)

`Boolean` 👉 `true` *ou* `false`

`null`

`undefined`

`Number`

`String`

Array (Matriz)

```
let meuArray = new Array("Valor 1", "Valor 2", "Valor 3");

// OU do jeito abaixo...

let meuArray = ["Valor 1", "Valor 2", "Valor 3"];

// Lembrando que o índice dos arrays em JS começam em 0 (meuArray[0])
```

Objetos

```
const meuObjeto = {
  nome: "Lucas",
  idade: 32,
};

console.log(meuObjeto.nome); // Lucas
console.log(meuObjeto.idade + " anos"); // 32 anos
```

Declaração em bloco

Uma declaração em bloco é utilizada para agrupar declarações.
O bloco é delimitado por um par de chaves:

```
{
  declaracao_1;
  declaracao_2;
  .
  .
  .
  declaracao_n;
}

// "blocos" assim, servem para if, else, for, while, function e etc
```

Declarações condicionais

```
if (condicao) {
  declaracao_1;
} else if (condicao_2) {
  declaracao_2;
} else if (condicao_n) {
  declaracao_n;
} else {
  declaracao_final;
}
```

Valores avaliados como falsos:

`false, undefined, null, 0, NaN e string vazia ("")`

Declaração switch

```
switch (expressao) {
  case rotulo_1:
    declaracoes_1
    [break;]
  case rotulo_2:
    declaracoes_2
    [break;]
    ...
  default:
    declaracoes_padrao
    [break;]
}
```

Estruturas de Repetição (loop)

```
for (let passo = 0; passo < 5; passo++) {
  console.log(`Andou ${passo} passos`);
}
```

```
let passo = 0;
while (passo < 5) {
  console.log(`Andou ${passo} passos`);
  passo++;
}
```

Use **`break`** para parar a execução de um loop ou continue para pular para o próximo passo

Declaração de Funções em JS

A definição da função *(também chamada de declaração de função)* consiste no uso da palavra chave function, seguida por:

- Nome da Função.
- Lista de argumentos para a função, entre parênteses e separados por vírgulas.
- Declarações JavaScript que definem a função, entre chaves { }.

```
function square(numero) {
  return numero * numero;
}

// e para executar essa função
square(5); // resultado = 25
```

Declaração de Funções em JS

Também é possível declarar uma função em JavaScript destas outras maneiras:

```
// alternativa 1
const square = function (numero) {
  return numero * numero;
};

// alternativa 2 (funções curtas)
const square = numero => numero * numero;
```

As variáveis definidas no interior de uma função não podem ser acessadas de nenhum lugar fora da função, porque a variável está definida apenas no escopo da função.

Operadores Básicos

- = atribuição
- * multiplicação
- / divisão
- - subtração
- + soma
- ++ incremento
- -- decremento
- == comparação
- === comparação estrita
- != diferença
- !== diferença estrita
- > e >= maior, maior igual
- < e <= menor, menor igual

Operador spread *(ES6)*

O operador **spread** permite que uma expressão seja expandida em locais onde são esperados vários argumentos *(para chamadas de função)* ou vários elementos *(para arrays)*.

```
let partes = ["ombro", "joelhos"];
let musica = ["cabeca", ...partes, "e", "pés"];
```

Definições de Classes *(JS Moderno - ES6)*

Em JavaScript, podemos fazer uma Classe (OOP):

```
class Employee {
  constructor() {
    this.name = "";
    this.dept = "general";
  }
  getMyName() {
    return this.name;
  }
  static builder() {
    return new Employee();
  }
}
```

Comparativo com o Java:

```
public class Employee {
  public String name = "";
  public String dept = "general";
  public String getMyName() {
    return this.name;
  }
}
```

Outras questões que vocês podem pesquisar:

- **Números** e funções **Math**:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math#static_methods
- **Datas** e o objeto **Date**:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
- Expressões Regulares (***RegExp***):
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp
- Métodos de array (*push*, *splice*, *slice*, *pop*, *concat*, *map*, *filter e etc)*:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#instance_methods

LET'S CODE!

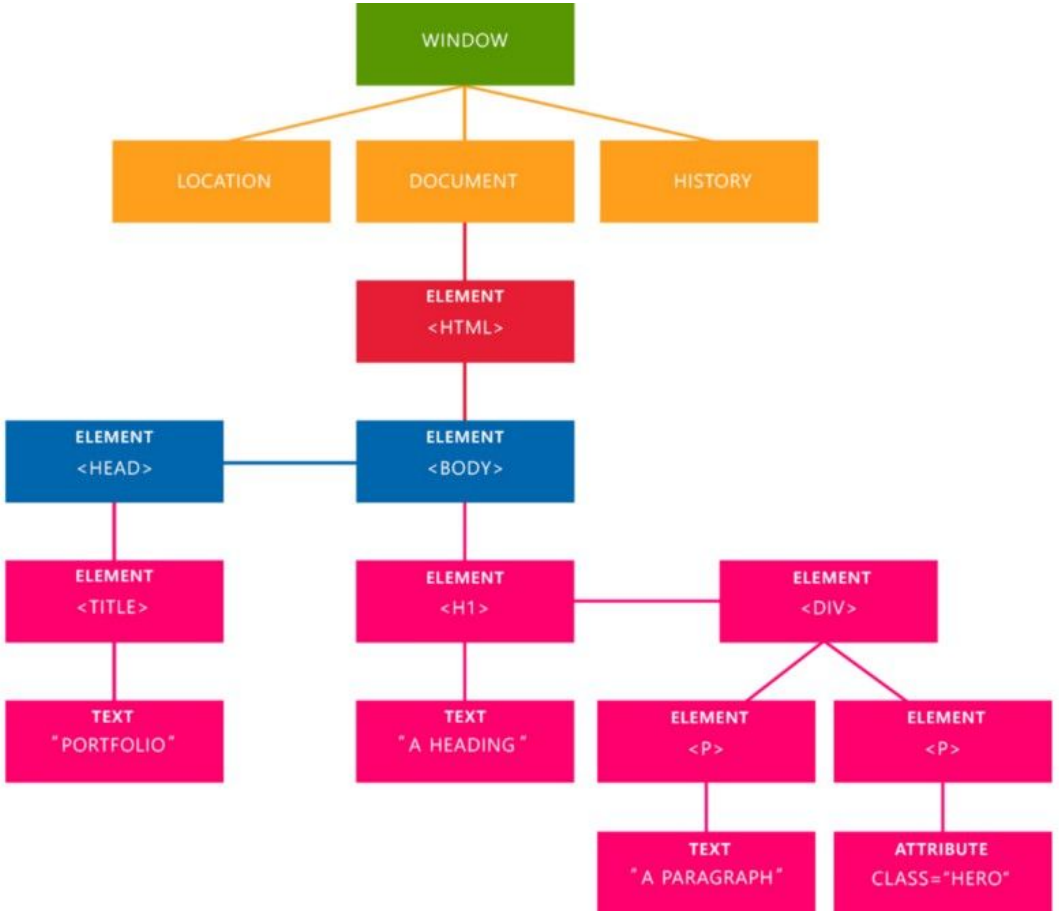
🤘

DOM: sua página no mundo JavaScript

Document Object Model

- O **DOM** *(Document Object Model)* é uma interface que representa como os documentos HTML são lidos pelo seu browser
- Após o browser ler seu documento HTML, ele cria um objeto que faz uma representação estruturada do seu documento e define meios de como essa estrutura pode ser acessada
- Essa estrutura pode ser acessada através da variável global `document`.
- Nós podemos acessar e manipular o **DOM** com JavaScript, é a forma mais fácil e usada de interação
- O termo *"documento"* é frequentemente utilizado em referências à nossa página. No mundo front-end, documento e página são sinônimos.

Representação do DOM pelo navegador



Métodos do DOM *(mais comuns)*

O **DOM** possui muitos métodos, são eles que fazem a ligação entre os nodes (elementos) e os eventos.

getElementById 👉 obtém um elemento através de seu ID

```
const myEl = document.getElementById("start");
```

getElementsByClassName 👉 obtém um ou vários elementos através de sua classe do CSS

```
const myContainer = document.getElementsByClassName("container");
```

Esse método retorna um ***HTMLCollection*** de todos elementos que estiverem contendo o nome da classe passada.

Métodos do DOM *(mais comuns)*

getElementsByTagName 👉 obtém um ou vários elementos através da TAG usada

```
const buttons = document.getElementsByTagName("button");
```

querySelector 👉 obtém um elemento através de seu seletor CSS

```
const resetButton = document.querySelector("form #reset");
```

querySelectorAll 👉 obtém vários elementos através de seus seletores CSS

```
const myButtons = document.querySelectorAll("header .button");
```

Percorrendo elementos do DOM

Após obter sua coleção de elementos, em especial seleções que retornem ***HTMLCollection*** pode ser necessário acessar todos de uma só vez, para isso será necessário um *loop*:

```
const fakeImages = document.querySelectorAll(".fake-image");
for (var i = 0; i < fakeImages.length; i++) {
  fakeImages[i].style.border = "1px solid red";
}
```

Eventos DOM

Após determinar o elemento alvo e obter com um dos métodos seletores, será possível adicionar **eventos de interação** a estes mesmos elementos.

Forma simples (não indicada):

```
const meuBotao = documento.getElementById("meuBotao");
meuBotao.onclick = function (event) {
  alert("Cliquei neste botão.");
};
```

Forma indicada:

```
meuBotao.addEventListener("click", function (event) {
  alert("Cliquei neste botão.");
});
```

Eventos mais comuns

- ***oninput*** 👉 quando um elemento input tem seu valor modificado
- ***onclick*** 👉 quando ocorre um click com o mouse
- ***onkeypress*** 👉 quando pressionar e soltar uma tecla
- ***onkeydown*** 👉 quando pressionar uma tecla
- ***onkeyup*** 👉 quando soltar uma tecla
- ***onblur*** 👉 quando um elemento perde foco
- ***onfocus*** 👉 quando um elemento ganha foco
- ***onchange*** 👉 quando um *input*, *select* ou *textarea* tem seu valor alterado
- ***onload*** 👉 quando a página é carregada
- ***onunload*** 👉 quando a página é fechada
- ***onsubmit*** 👉 disparado antes de submeter o formulário (útil para realizar validações)

Criando elementos

Após interagirmos com elementos já presentes em tela e "monitorar" os seus eventos de ação *(ex: click de um botão)*, o próximo passo natural seria utilizarmos JS para **adicionar novos elementos em tela**:

```
function adicionaItem() {
  // criando um novo elemento do tipo "<li>" e
  // definindo seu conteúdo
  const novoItem = document.createElement("li");
  novoItem.innerHTML = "Conteúdo novo item";

  // recuperando um elemento id="lista" (tipo <ul>)
  // para adicionar um novo "child" dentro dele
  const lista = document.querySelector("#lista");
  lista.appendChild(novoItem);
}
```

Criando elementos

Inicialmente, a criação de elementos novos de HTML em JS para dispormos em tela é relativamente "fácil".

Já sabemos que o objeto "***document***" é o mandatário das principais funções de JS relacionados a manipulação de HTML com JS.

Logo a função indicada para criar novos elementos no DOM e posteriormente adicionar em tela será "**createElement**".

Por exemplo, se quisermos criar uma ***<DIV>*** nova e *setarmos* um conteúdo "rico" para ela:

```
const novaDiv = document.createElement("div");
novaDiv.innerHTML = "<strong>Podemos também ter tags aqui</strong>";
```

Criando elementos

Vale observar *(continuando)* que todo elemento criado encontra-se apenas em "memória virtual", podemos armazenar um novo elemento do DOM/HTML em uma variável mas enquanto não "adicionarmos" o mesmo em tela ele nunca será exibido para o usuário. O caminho mais fácil para trazer oficialmente este item em tela é usando a função "**appendChild**".

Porém é muito importante perceber que a função ***appendChild*** só funciona em uma chamada a partir do "elemento pai" que irá receber os novos elementos "filhos" em sua hierarquia e dispor em tela para o usuário. Logo antes de adicionarmos um novo elemento devemos primeiro estabelecer um "elemento alvo" para receber o mesmo:

```
// a ideia abaixo supõe que tenhamos um "painel" genérico
// com id="container" a fim de receber novos elementos
// aí recuperarmos o "pai" em uma variável comum de JS
// e depois podemos acionar "appendChild" para finalizar
const container = document.querySelector("#container");
container.appendChild(novaDiv);
```

Removendo elementos

Outra manipulação básica de elementos é remover elementos de tela. Independente do elemento existir oficialmente escrito no arquivo .html OU ter sido criado por JavaScript, a primeira análise a ser feita quanto a exclusão de elementos é se vamos apenas "esconder" ou se iremos realmente mover *(para sempre)*.

Primeiro recuperarmos o elemento, vamos supor um elemento com a classe ***.mensagem-erro*** que precisa sumir da tela:

```
const mensagemErro = document.querySelector(".mensagem-erro");
```

Depois para escondermos este elemento de tela *(remoção visual)* podemos mudar apenas o estado do estilo de "disposição" do elemento:

```
mensagemErro.style.display = "none";
```

Removendo elementos

Na sequência outra opção "mais radical" seria realmente remover o elemento do HTML *(para sempre)*, neste caso a função mais adequada seria a ***removeChild***, que assim como a *appendChild* deve ser acionada a partir do elemento PAI que contém o filho a ser "removido":

```
// primeiro recuperarmos o "elemento filho" a ser removido
const mensagemErro = document.querySelector(".mensagem-erro");

// também "selecionamos" o "elemento pai" que perderá seu "filho"
const container = document.querySelector("#container");

// por fim podemos remover a "mensagemErro" de dentro e seu local pai
container.removeChild(mensagemErro);
```

KEEP ROCKIN!

🤘

E finalmente...

AJAX!

AJAX

- *Asynchronous Javascript and XML*, ou **AJAX** é originalmente o uso metodológico de tecnologias como Javascript e XML, para tornar páginas Web mais interativas para o usuário, utilizando-se principalmente de solicitações assíncronas de informações
- Apesar do nome, a utilização de XML não é obrigatória sendo que **JSON** é amplamente mais usado atualmente

E comofas?

=> GET

```
// Requisição padrão / clássica

var minhaRequisicao = new XMLHttpRequest();
minhaRequisicao.responseType = "json"; // opcional...
minhaRequisicao.onload = function () {
  if (minhaRequisicao.status === 200) {
    // aqui vem o retorno do carregamento assíncrono
    console.log(minhaRequisicao.response);
  }
};
minhaRequisicao.open("GET", "/minhaUrlNoServidor");
minhaRequisicao.send();
```

E comofas?

=> POST

```
// Requisição clássica com POST

var formData = new FormData();
formData.append("nome", "Lee");
formData.append("sobrenome", "Souza");
var minhaRequisicao = new XMLHttpRequest();
minhaRequisicao.onload = function () {
  if (minhaRequisicao.status === 200) {
    // aqui vem o retorno do carregamento assíncrono
    console.log(minhaRequisicao.response);
  }
};
minhaRequisicao.open("POST", "/minhaUrlNoServidor");
minhaRequisicao.send(formData); // passa o objeto com dados de post aqui
```

E comofas?

=> GET

```
// Requisição "moderna":

fetch("/minhaUrlNoServidor")
  .then(response => {
    return response.text(); // response.json()
  })
  .then(data => {
    console.log(data); // retorno de texto...
  });
```

E comofas?

=> POST

```
// Requisição "moderna" com POST:

var formData = new FormData();
formData.append("nome", "Lee");
formData.append("sobrenome", "Souza");
fetch("/minhaUrlNoServidor", { method: "POST", body: formData })
  .then(response => {
    return response.text(); // response.json()
  })
  .then(data => {
    console.log(data); // retorno de texto...
  });
```

SOLO FINAL 🤘

Bibliotecas Maneiras *(para checar)*

- **Axios**: https://github.com/axios/axios
- **XR**: https://github.com/radiosilence/xr
- **jQuery**: https://jquery.com/
- **Parsley**: http://parsleyjs.org/
- **YUP**: https://github.com/jquense/yup
- **ZOD**: https://github.com/colinhacks/zod

E por fim...

Nosso 2º Exercício!

valendo nota evidentemente

Lembram do CHAT html & css que iniciamos no outro exercício?

Aguardem novidades... 😈

obrigado 🚀