GEX613 – Programação II

DOM - Document Object Model

1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. Giancarlo Salton

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Document Object Model

HTML Dinâmico

DOM Nodes

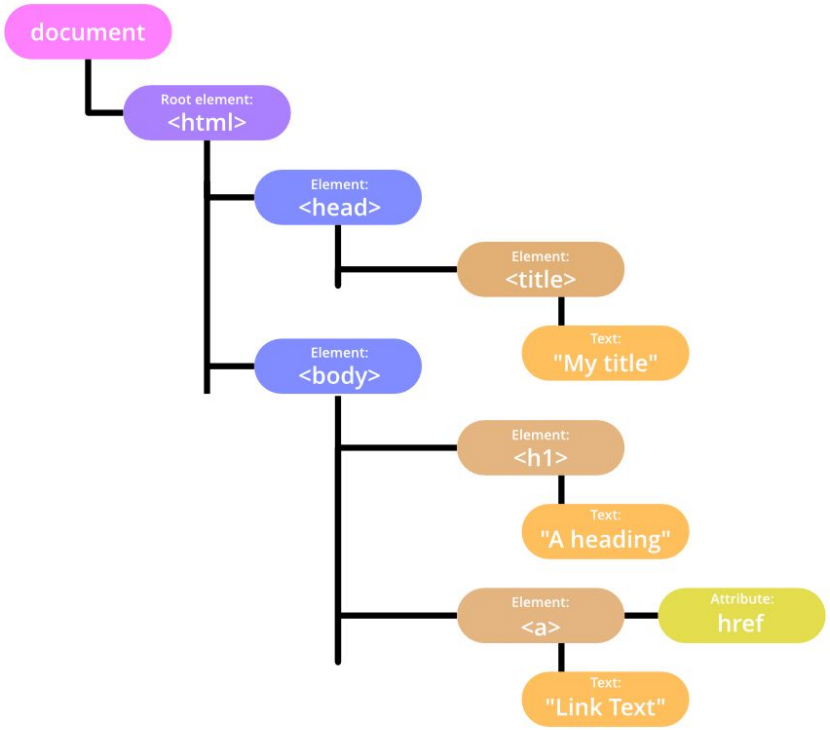
jQuery

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Document **O**bject **M**odel

DOM

- Quando uma página web é carregada, o navegador cria um Modelo de Objeto de Documento (DOM) da página.
- O modelo DOM do HTML é construído como uma árvore de objetos.



O que é HTML DOM?

- DOM do HTML faz parte do DOM da W3C (World Wide Web Consortium).

- O padrão DOM da W3C é dividido em 3 partes diferentes:
  - ✔ DOM Core - modelo padrão para todos os tipos de documentos
  - ✔ DOM XML - modelo padrão para documentos XML
  - ✔ DOM HTML - modelo padrão para documentos HTML

O que é HTML DOM?

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- O DOM do HTML é um modelo de objeto padrão e interface de programação para HTML. Ele define:
  - ✔ Os elementos do HTML como objetos
  - ✔ As propriedades de todos os elementos do HTML
  - ✔ Os métodos para acessar todos os elementos do HTML
  - ✔ Os eventos para todos os elementos do HTML

- Em outras palavras, o DOM do HTML é um **padrão sobre como obter, alterar, adicionar ou excluir elementos do HTML**.

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

HTML “Dinâmico”

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- O JavaScript pode:

  - ✔ Alterar todos os elementos e atributos do HTML na página
  - ✔ Alterar todos os estilos CSS na página
  - ✔ Remover elementos e atributos do HTML existentes
  - ✔ Adicionar novos elementos e atributos do HTML
  - ✔ Reagir a todos os eventos do HTML existentes na página
  - ✔ Criar novos eventos do HTML na página

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- No DOM, todos os elementos do HTML são definidos como objetos.
- A interface de programação é o conjunto de propriedades e métodos de cada objeto.
  - ✔ Uma **propriedade** é um valor que você pode obter ou definir.
  - ✔ Um **método** é uma ação que você pode fazer
- O objeto `document` é o “dono” de todos os objetos contidos na página.

Encontrando elementos HTML

- Frequentemente, com o JavaScript, você deseja manipular elementos HTML. Para fazer isso, você precisa encontrar os elementos primeiro. Você pode encontrar elementos HTML:

  - ✔ por id
  - ✔ por nome de tag
  - ✔ por nome de classe
  - ✔ por coleções de objetos HTML (isto é, todos os objetos de um mesmo tipo)

Encontrando elementos no DOM

```javascript
// Encontrando um elemento pelo ID
var elementoPorId = document.getElementById("meuId");


// Encontrando elementos pelo nome da tag
var elementosPorTag = document.getElementsByTagName("p");


// Encontrando elementos pelo nome da classe
var elementosPorClasse = document.getElementsByClassName("minhaClasse");


// Encontrando elementos por coleções de objetos HTML
// Exemplo: todos os formulários da página
var colecoesDeFormularios = document.forms;
```

Alterando elementos HTML

- Para alterar o conteúdo de um elemento HTML, use esta sintaxe:
  - ✔ `document.getElementById(id).innerHTML = novoHTML;`

- Para alterar o valor de um atributo HTML, use esta sintaxe:
  - ✔ `document.getElementById(id).attribute = novoValor;`

Exemplo: Alterando valor do atributo

```
<!DOCTYPE html>
<html>
<body>
<img id="image" src="smiley.gif" width="160" height="120">

<script>
   document.getElementById("image").src="landscape.jpg";
</script>

<p>The original image was smiley.gif, but the script changed it to landscape.jpg</p>

</body>
</html>
```

Alterando o estilo (CSS) do HTML

- Para mudar o estilo de um elemento HTML:

  ✔ `document.getElementById("id").style.color = "red";`

Exemplo: Alterando a visibilidade de um parágrafo

```
<p id="p1">
   This is some text.
   This is a line of text.
</p>
<p>This is another line of text.</p>

<input type="button" value="Hide text"
   onclick="document.getElementById('p1').style.visibility='hidden'">
<input type="button" value="Show text"
   onclick="document.getElementById('p1').style.visibility='visible'">
```

- Para adicionar um manipulador de eventos em um elemento:

  ✔ `element.addEventListener(event, function);`

- A função também pode ser pode ser uma *arrow function* :

  ✔ `element.addEventListener(event, () ⇒ {});`

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Exemplo: Adicionando método click em um botão

```
<!-- Botão com um ID para fácil acesso via JavaScript -->
<button id="meuBotao">Clique-me</button>

<!-- Incluindo o JavaScript no final do corpo para garantir que o DOM esteja carregado -->
<script>
    let botao = document.getElementById("meuBotao");
    // Adiciona o evento de clique ao botão
    botao.addEventListener("click", function() {
    alert("Botão clicado!");
    });
</script>
```

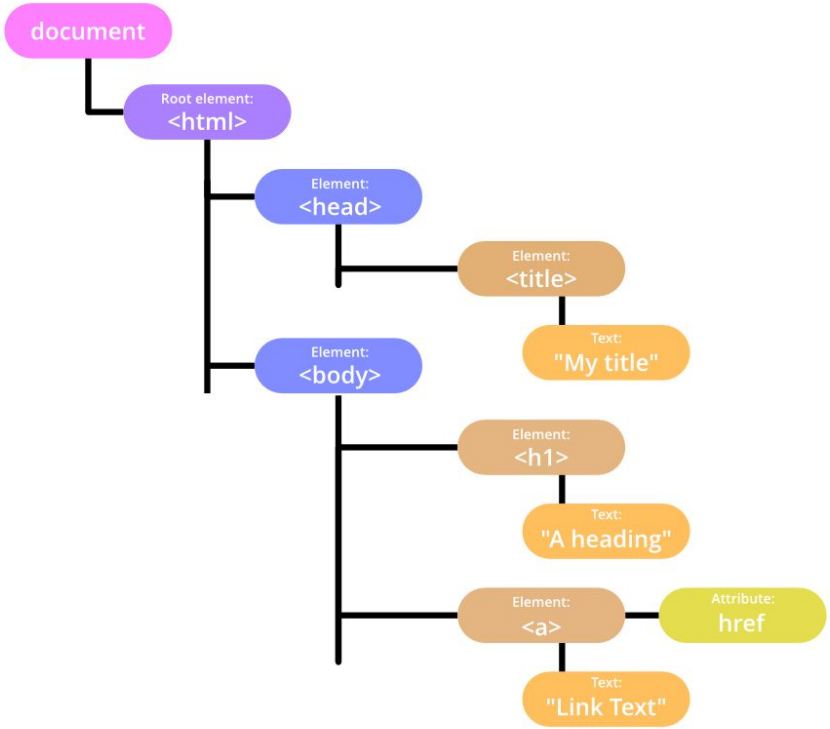
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

DOM Nodes

DOM Nodes

- De acordo com o padrão DOM (Modelo de Objeto de Documento) da W3C (World Wide Web Consortium) para HTML, tudo em um documento HTML é um nó:

  - ✔ O documento inteiro é um document node
  - ✔ Cada elemento HTML é um element node
  - ✔ O texto dentro dos elementos HTML são text nodes.
  - ✔ Cada atributo HTML é um attribute node.
  - ✔ Todos os comentários são comment nodes.



Navegando entre os nodos

- Você pode usar as seguintes propriedades de nós para navegar entre os nós com JavaScript:

  - ✔ `parentNode` (nó pai)
  - ✔ `childNodes[nodeNumber]` (filhos do nó [número do nó])
  - ✔ `firstChild` (primeiro filho)
  - ✔ `lastChild` (último filho)
  - ✔ `nextSibling` (próximo nó irmão)
  - ✔ `spreviousSibling` (nó irmão anterior)

Exemplo: Acessando um node filho

```
<body>
    <p id="intro">Hello World!</p>
    <script>
        let txt = document.getElementById("intro").childNodes[0].nodeValue;
        console.log(txt + " Repeated");
    </script>
</body>
```

Para acessar o primeiro node, também poderíamos utilizar

```
let txt = document.getElementById("intro").firstChild.nodeValue;
```

Exemplo: Acessando um node irmão

```
<p>Menu</p>
<ol id="myMenu">
   <li id="first"><b>Fresh</b><i>Coffee </i> </li>
   <li>Tea</li>
</ol>
<script>
   let txt1 = document.getElementById("first").firstChild;
   console.log(txt1);
   let txt2 = txt1.nextSibling;
   console.log(txt2);
</script>
```

Para acessar o próximo irmão, também poderíamos utilizar

```
let txt2 = document.getElementById("first").childNodes[1];
```

Root Nodes

- Existem dois nodes especiais que permite acesso ao documento completo
  - ✔ `document.documentElement` – o documento inteiro
  - ✔ `document.body` – o corpo (*body*) do documento

nodeName

- A propriedade `nodeName` indica o nome de um nó.

  - ✔ `nodeName` é somente leitura.
  - ✔ `nodeName` de um nó de elemento é o mesmo que o nome da tag em MAIÚSCULAS.
  - ✔ `nodeName` de um nó de atributo é o nome do atributo.
  - ✔ `nodeName` de um nó de texto é sempre `#text`
  - ✔ `nodeName` de um nó de documento é sempre `#document`.
  - ✔ `nodeName` de um nó de comentário é sempre `#comment`.

- Exemplo:

```
let x = document.body.nodeName;
// x == `BODY`
```

nodeValue

- A propriedade `nodeValue` indica o nome de um nó.

  - ✔ `nodeValue` é leitura/escrita.
  - ✔ `nodeValue` de um nó de elemento é `undefined`/`null`
  - ✔ `nodeValue` de um nó de atributo é o valor do atributo.
  - ✔ `nodeValue` de um nó de texto é sempre o conteúdo de texto
  - ✔ `nodeValue` de um nó de documento é sempre `undefined`/`null`.
  - ✔ `nodeValue` de um nó de comentário é sempre conteúdo de texto do comentário

Criando novos elementos HTML

- O DOM pode ser manipulado adicionando e anexando elementos à árvore do DOM, removendo elementos da árvore do DOM ou substituindo elementos.

- Para adicionar um novo elemento ao DOM do HTML, você deve primeiro criar o elemento (nó de elemento) e, em seguida, anexá-lo a um elemento existente:
  - ✔ `appendChild()` – insere o novo elemento como o último filho do elemento
  - ✔ `insertBefore()` – insere o novo elemento antes do elemento, tornando-o filho do elemento inserido

Exemplo: appendChild

```
<body>
    <div id="div1">
        <p id="p1">This is a paragraph.</p>
        <p id="p2">This is another paragraph.</p>
    </div>

    <script>
        let para = document.createElement("p");
        let node = document.createTextNode("This is new.");
        para.appendChild(node);

        let element = document.getElementById("div1");
        element.appendChild(para);
    </script>
</body>
```

Exemplo: insertBefore

```
<div id="div1">
   <p id="p1">This is a paragraph.</p>
   <p id="p2">This is another paragraph.</p>
</div>

<script>
   let para = document.createElement("p");
   let node = document.createTextNode("This is new.");
   para.appendChild(node);
   let child = document.getElementById("p1");
   let div = document.getElementById(”div1");
   div.insertBefore(para, child); // para vai antes de child
</script>
```

Removendo elementos HTML

- Para remover um elemento, precisamos primeiro conhecer o nodo pai:

```
<div id="div1">
   <p id="p1">This is a paragraph.</p>
   <p id="p2">This is another paragraph.</p>
</div>

<script>
   let parent = document.getElementById("div1");
   let child = document.getElementById("p1");
   parent.removeChild(child);
</script>
```

Substituindo elementos HTML

- Para substituir um elemento, precisamos conhecer o nodo pai e utilizar o método `replaceChild()`:

```
<div id="div1">
   <p id="p1">This is a paragraph.</p>
   <p id="p2">This is another paragraph.</p>
</div>

<script>
   let para = document.createElement("p");
   let node = document.createTextNode("This is new.");
   para.appendChild(node);
   let parent = document.getElementById("div1");
   let child = document.getElementById("p1");
   parent.replaceChild(para, child); // para substitui child
</script>
```

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

jQuery

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- jQuery é uma biblioteca JavaScript.
- Grande coleção de utilidades e funcionalidades comuns.
- Ajuda a tornar a vida dos desenvolvedores mais fácil e a codificação mais eficiente.
  - ✔ "Escreva menos, faça mais"
- Existem muitas bibliotecas JavaScript, mas atualmente o jQuery é a mais utilizada
- Contém códigos para manipular HTML, CSS, adicionar/remover *eventListeners*, efeitos e animações, entre outros

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

Selecting Elements by ID

```
1 | $( "#myId" ); // Note IDs must be unique per page.
```

Selecting Elements by Class Name

```
1 | $( ".myClass" );
```

Selecting Elements by Attribute

```
1 | $( "input[name='first_name']" );
```

Selecting Elements by Compound CSS Selector

```
1 | $( "#contents ul.people li" );
```

Selecting Elements with a Comma-separated List of Selectors

```
1 | $( "div.myClass, ul.people" );
```

- *Getters & Setters*

```
// The .html() method sets all the h1 elements' html to be "hello world":
$( "h1" ).html( "hello world" );
```

```
// The .html() method returns the html of the first h1 element:
$( "h1" ).html();
// > "hello world"
```

- *Chaining*

```
// Changing the HTML of an element.
$( "#myDiv p:first" ).html( "New <strong>first</strong> paragraph!" );
```

Movendo e clonando elementos

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- `.insertAfter()`
- `.insertBefore()`
- `.appendTo()`
- `.prependTo()`

```
// Moving elements using different approaches.

// Make the first list item the last list item:
var li = $( "#myList li:first" ).appendTo( "#myList" );

// Another approach to the same problem:
$( "#myList" ).append( $( "#myList li:first" ) );

// Note that there's no way to access the list item
// that we moved, as this returns the list itself.
```

```
// Making a copy of an element.

// Copy the first list item to the end of the list:
$( "#myList li:first" ).clone().appendTo( "#myList" );
```

Criando elementos

```
1 | // Creating new elements from an HTML string.
2 | $( "<p>This is a new paragraph</p>" );
3 | $( "<li class=\"new\">new list item</li>" );
```

```
1 | // Creating a new element with an attribute object.
2 | $( "<a/>", {
3 |     html: "This is a <strong>new</strong> link",
4 |     "class": "new",
5 |     href: "foo.html"
6 | });
```

- Apenas criar os elementos não vai adicioná-los à página!

```
1 | // Creating and adding an element to the page at the same time.
2 | $( "ul" ).append( "<li>list item</li>" );
```

Criando elementos

```jquery
// Getting a new element on to the page.

var myNewElement = $( "<p>New element</p>" );

myNewElement.appendTo( "#content" );

myNewElement.insertAfter( "ul:last" ); // This will remove the p from #content!

$( "ul" ).last().after( myNewElement.clone() ); // Clone the p so now we have two.
```

- Antes tínhamos que fazer

```javascript
let para = document.createElement("p");
let node = document.createTextNode("This is new.");
para.appendChild(node);
let element = document.getElementById("div1");
element.appendChild(para);
```

Manipulando elementos

```
// Manipulating a single attribute.
$( "#myDiv a:first" ).attr( "href", "newDestination.html" );
```

```
// Manipulating multiple attributes.
$( "#myDiv a:first" ).attr({
    href: "newDestination.html",
    rel: "nofollow"
});
```

```
// Using a function to determine an attribute's new value.
$( "#myDiv a:first" ).attr({
    rel: "nofollow",
    href: function( idx, href ) {
        return "/new/" + href;
    }
});

$( "#myDiv a:first" ).attr( "href", function( idx, href ) {
    return "/new/" + href;
});
```

Removendo elementos

```
.remove()
```

- este método retorna o elemento e depois remove este elemento da página, incluindo dados e *eventListeners*

```
.detach()
```

- este método retorna o elemento e depois remove este elemento da página, mantendo dados e *eventListeners*

Iteração sobre elementos pais

```html
<div class="grandparent">
    <div class="parent">
        <div class="child">
            <span class="subchild"></span>
        </div>
    </div>
    <div class="surrogateParent1"></div>
    <div class="surrogateParent2"></div>
</div>
```

- .parent()
- .parents()
- .parentsUntil()
- .closest()

```javascript
// Selecting an element's direct parent:

// returns [ div.child ]
$( "span.subchild" ).parent();

// Selecting all the parents of an element that match a given selector:

// returns [ div.parent ]
$( "span.subchild" ).parents( "div.parent" );

// returns [ div.child, div.parent, div.grandparent ]
$( "span.subchild" ).parents();

// Selecting all the parents of an element up to, but *not including* the selector:

// returns [ div.child, div.parent ]
$( "span.subchild" ).parentsUntil( "div.grandparent" );

// Selecting the closest parent, note that only one parent will be selected
// and that the initial element itself is included in the search:

// returns [ div.child ]
$( "span.subchild" ).closest( "div" );

// returns [ div.child ] as the selector is also included in the search:
$( "div.child" ).closest( "div" );
```

Iteração sobre elementos filhos

```html
<div class="grandparent">
    <div class="parent">
        <div class="child">
            <span class="subchild"></span>
        </div>
    </div>
    <div class="surrogateParent1"></div>
    <div class="surrogateParent2"></div>
</div>
```

- .chilren()
- .find()

```javascript
// Selecting an element's direct children:

// returns [ div.parent, div.surrogateParent1, div.surrogateParent2 ]
$( "div.grandparent" ).children( "div" );

// Finding all elements within a selection that match the selector:

// returns [ div.child, div.parent, div.surrogateParent1, div.surrogateParent2 ]
$( "div.grandparent" ).find( "div" );
```

Iteração sobre elementos irmãos

```html
<div class="grandparent">
    <div class="parent">
        <div class="child">
            <span class="subchild"></span>
        </div>
    </div>
    <div class="surrogateParent1"></div>
    <div class="surrogateParent2"></div>
</div>
```

- .prev()
- .next()
- .siblings()
- .nextAll()
- .nextUntil()
- .prevAll()
- .prevUntil()

```javascript
// Selecting a next sibling of the selectors:

// returns [ div.surrogateParent1 ]
$( "div.parent" ).next();

// Selecting a prev sibling of the selectors:

// returns [] as No sibling exists before div.parent
$( "div.parent" ).prev();

// Selecting all the next siblings of the selector:

// returns [ div.surrogateParent1, div.surrogateParent2 ]
$( "div.parent" ).nextAll();

// returns [ div.surrogateParent1 ]
$( "div.parent" ).nextAll().first();

// returns [ div.surrogateParent2 ]
$( "div.parent" ).nextAll().last();

// Selecting all the previous siblings of the selector:

// returns [ div.surrogateParent1, div.parent ]
$( "div.surrogateParent2" ).prevAll();

// returns [ div.surrogateParent1 ]
$( "div.surrogateParent2" ).prevAll().first();

// returns [ div.parent ]
$( "div.surrogateParent2" ).prevAll().last();
```

Eventos

```
// Event setup using a convenience method
$( "p" ).click(function() {
    console.log( "You clicked a paragraph!" );
});
```

```
// Equivalent event setup using the `.on()` method
$( "p" ).on( "click", function() {
    console.log( "click" );
});
```

Eventos

```
1 // Event setup using a convenience method
2 $( "p" ).click(function() {
3     console.log( "You clicked a paragraph!" );
4 });
```

```
1 // Equivalent event setup using the `.on()` method
2 $( "p" ).on( "click", function() {
3     console.log( "click" );
4 });
```

Perceba que o efeito é o mesmo, setando diretamente `click` ou utilizando o método `on` e passando o tipo do evento como parâmetro.

Many events, but only one event handler

```
1 | // When a user focuses on or changes any input element,
2 | // we expect a console message bind to multiple events
3 | $( "div" ).on( "mouseenter mouseleave", function() {
4 |     console.log( "mouse hovered over or left a div" );
5 | });
```

Many events and handlers

```
 1 | $( "div" ).on({
 2 |     mouseenter: function() {
 3 |         console.log( "hovered over a div" );
 4 |     },
 5 |     mouseleave: function() {
 6 |         console.log( "mouse left a div" );
 7 |     },
 8 |     click: function() {
 9 |         console.log( "clicked on a div" );
10 |     }
11 | });
```