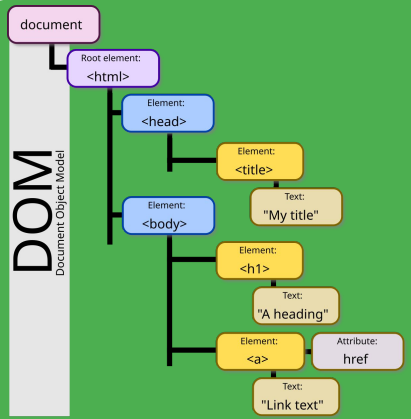
Desenvolvimento WEB II
Prof. Ernani Gottardo

DOM - Document Object Model



DOM

- Quando uma página da Web é carregada, o navegador cria um **DOCUMENT OBJECT MODEL** (DOM).
- DOM fornece uma **representação estruturada** do documento como uma árvore.
- O DOM define **métodos** que permitem acesso à árvore, para que eles possam alterar a estrutura, estilo e conteúdo do documento.
- O DOM é uma **interface entre a linguagem (ex JS)** e os elementos HTML
    - DOM é uma estrutura do HTML não do JS

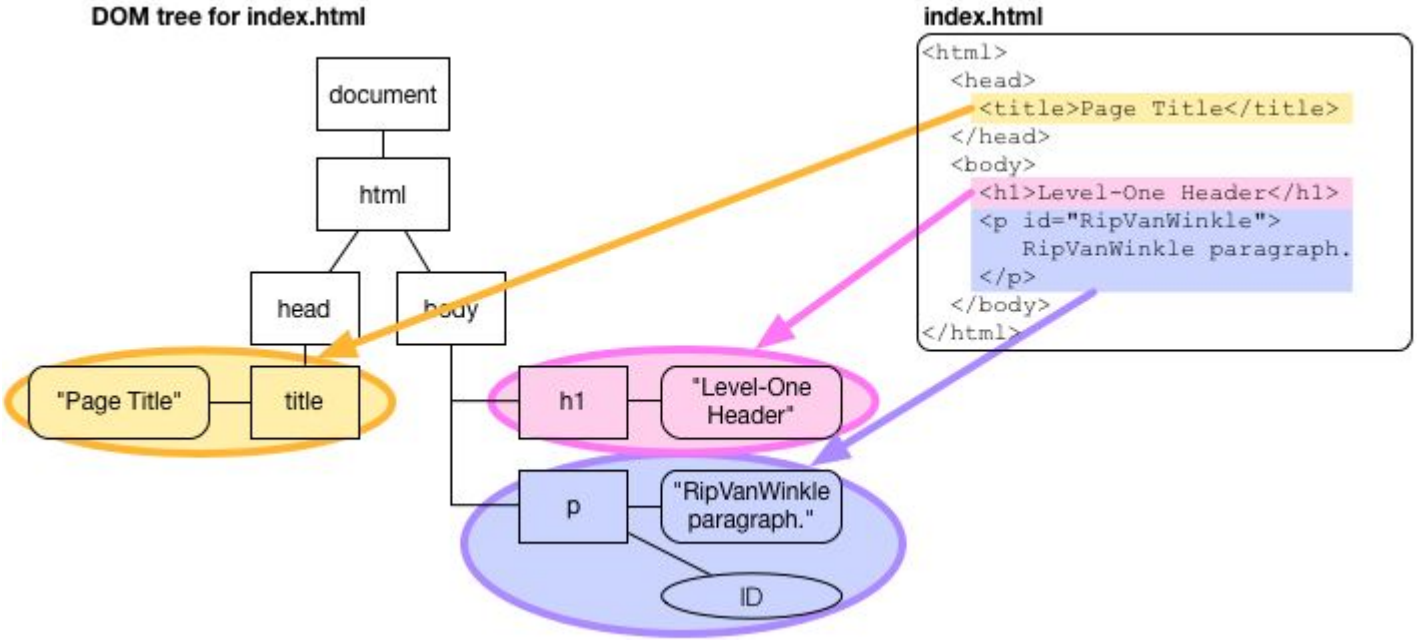
DOM

- Através da estrutura do DOM é possível:
  - mudar todos os elementos HTML na página
  - alterar todos os atributos HTML na página
  - mudar todos os estilos CSS na página
  - remover elementos e atributos HTML existentes
  - adicionar novos elementos e atributos HTML
  - reagir a todos os eventos HTML existentes na página
  - criar novos eventos HTML na página

DOM

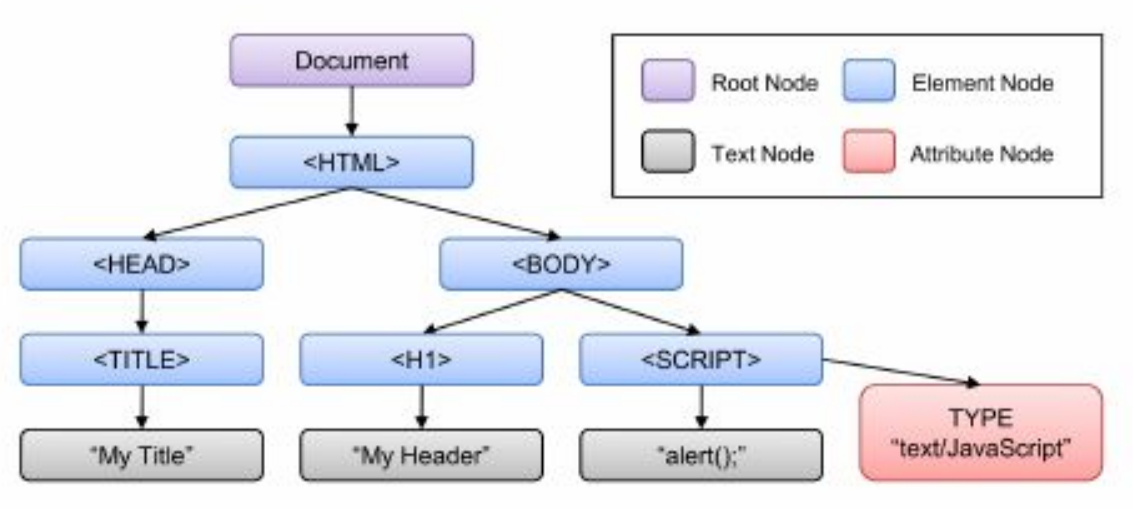
- Através da estrutura do DOM é possível:
  - mudar todos os elementos HTML na página
  - alterar todos os atributos HTML na página
  - mudar todos os estilos CSS na página
  - remover elementos e atributos HTML existentes
  - adicionar novos elementos e atributos HTML
  - reagir a todos os eventos HTML existentes na página
  - criar novos eventos HTML na página

Árvore DOM



Testar usando https://bioub.github.io/dom-visualizer/

DOM - tipos de nós(node)



Localizar elementos no DOM

- Frequentemente, com JavaScript, é necessário manipular elementos HTML/DOM.
- Para fazer isso, é necessário encontrar o(s) elemento(s) primeiro. Existem várias maneiras de encontrar elementos:
  - por **id**
  - por **nome de tag**
  - por **nome de classe**
  - por **seletores CSS**

Localizar elementos por id

- A maneira mais fácil de encontrar um elemento HTML no DOM é usando o **id** do elemento.
- Este exemplo encontra o elemento com id="intro":

```
<p id="intro">Localizar Elementos HTML pelo Id</p>
let element = document.getElementById("intro");
```

- Se o elemento for encontrado, o método retornará o elemento como **um** objeto (na variável ***element***).
- Se o elemento não for encontrado, *element* conterá null.

Localizar elementos por nome da tag

- JS permite localizar todos os elementos de determinada tag
- Este exemplo encontra o elemento com tag="p":

```
<p id="intro">Localizar Elementos HTML pelo Id</p>
<p> Segundo Parágrafo </p>
let element = document.getElementByTagName("p");
```

- Se o elemento for encontrado, o método retornará o(s) elemento(s) como **um array de** objetos (na variável ***element***).
- Se o elemento não for encontrado, *element* conterá null.

Localizar elementos por nome da classe

- JS permite localizar todos os elementos de determinada classe
- Este exemplo encontra o elemento com classe="dif":

```
<p class='dif' id="intro">Localizar Elementos HTML pelo Id</p>
<p class='dif'> Segundo Parágrafo </p>
<p> Terceiro Parágrafo </p>
let element = document.getElementByClassName("dif");
```

- Se o elemento for encontrado, o método retornará o(s) elemento(s) como **um array de** objetos (na variável ***element***).
- Se o elemento não for encontrado, *element* conterá null.

Localizar elementos por seletor CSS

- JS permite encontrar todos os elementos HTML que correspondem a um **seletor CSS** especificado (id, nomes de classes, tipos, atributos, valores de atributos, etc.)
- Este exemplo encontra o elemento com classe="dif":

```
<p class='dif' id="intro">Localizar Elementos HTML pelo Id</p>
<h2 class='dif'> Título Nível 2 </h2>
<p> Terceiro Parágrafo </p>
let element = document.querySelectorAll("p.dif");
```

- Se o elemento for encontrado, o método retornará o(s) elemento(s) como **um array de** objetos (na variável ***element***).
- Se o elemento não for encontrado, *element* conterá null.

Percorrer elementos selecionados

- Para os métodos que retornam um array de elementos, é possível percorrer o array com os elementos selecionados
- Exemplo

```
let element = document.getElementsByTagName("p");
for (let i = 0; i < element.length; i++) {
   document.write('<h1>'+ element[i].innerText+'</h1>');
}
```

Alterar elementos no DOM

Método	Descrição
element*.innerHTML = *new html content	Altera o conteúdo HTML de um elemento
element*.innerHTML = *new text content	Altera o conteúdo(texto) de um elemento
***element*.setAttribute(*attribute, value*)**	Altera o valor de um atributo de um elemento HTML
element*.style.*property* = *new style	Altera estilo de um elemento HTML

Exemplo manipulação do dom

```
<body>

    <p id="p1">Hello World!</p>

<script>
    let p1=document.getElementById("p1");
    p1.innerHTML = "New text!";
    p1.style.color = "red";
    p1.setAttribute("class","dif");//criar o css da classe
</script>

</body>
```

DOM - input propriedade value

- A propriedade **value** define ou retorna o valor do atributo value de um campo de texto.
- Exemplo de input

```
<input type='text' id='nome'>
```

- Recuperar o valor de um campo de texto:

```
let nome= document.getElementById('nome');
let n = nome.value;
```

- Alterar o valor de um campo de texto:

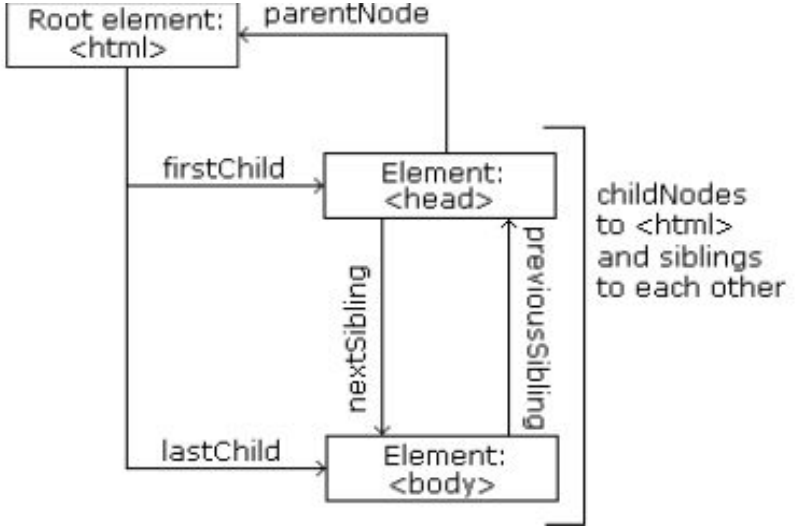
```
let nome= document.getElementById('nome');
nome.value='Paulo Silva';
```

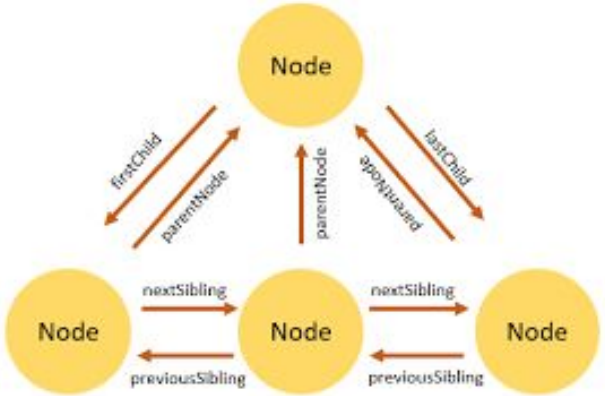
DOM - Relacionamento entre nós

- Em uma árvore DOM:
  - Nó topo é chamado de principal (root)
  - Todo nó possui um pai (exceto o root)
  - Um nó pode ter qualquer número de filhos(child)
  - Irmãos (siblings) são nós com o mesmo pai
  - Um nó sem filhos é chamado de folha(leaf)

DOM - navegar entre entre os nós

- childElementCount
- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling





DOM - navegar entre entre os nós

- HTML

```
<ul id="municipios">
   <li>Erechim</li>
   <li>Paulo Bento</li>
   <li>Aratiba</li>
 </ul>
```

- JS

```
let lista = document.getElementById('municipios');
 for (let i = 0; i < lista.childElementCount; i++) {
   console.log(lista.childNodes[i].innerText);
 }
```

DOM - navegar entre entre os nós

- Ao utilizar as funções de navegação pelos nós da árvore DOM deve-se observar que **espaços em branco, nova linha, comentários** (etc) são considerados elementos.

```
function clean(node) {
    for (var n = 0; n < node.childNodes.length; n++) {
        var child = node.childNodes[n];
        //types 1: element, 8:comentário, 3:texto
        if (
            child.nodeType === 8 ||
            (child.nodeType === 3 && !/\S/.test(child.nodeValue))
        ) {
            node.removeChild(child);
            n--;
        } else if (child.nodeType === 1) {
            clean(child);
        }
    }
}
```

DOM - excluir elemento

- O método remove() remove um elemento (ou nó) do documento
- Exemplo - html

```
<p id="demo">Parágrafo para remover.</p>
```

- Remover

```
const element = document.getElementById("demo");
element.remove();
```

DOM - incluir elemento/posição

```
<div id="div1">
  <p id="p1">Este é um parágrafo.</p>
  <p id="p2">Este é outro parágrafo</p>
</div>
```

- Para adicionar um novo elemento ao HTML DOM, em uma posição específica use **after ou before**:

```
let para = document.createElement("p");
let node = document.createTextNode("Novo Parágrafo depois");
para.appendChild(node);

let element = document.getElementById("p1");
element.after(para);
```

End