

DOM - Document Object Model

Disciplina: Desenvolvimento Web

Prof. Dr. Rafael Will M. de Araujo



Conteúdo

1 Introdução

2 Acessando elementos do HTML

Sumário

- O que é o DOM
- Como funciona a estrutura em árvore do DOM
- Como fazer buscas em elementos HTML
- Como manipular (criar e alterar) o DOM

DOM - *Document Object Model*

- DOM - “Modelo de Documento por Objetos”
- Quando uma página é carregada o navegador cria um modelo de objetos que representa a página (DOM).
- O DOM é construído como uma árvore de objetos:
 - ▷ Lembre-se que o HTML é basicamente um conjunto de marcadores aninhados.
 - ▷ Um marcador pode ter outros marcadores contidos nele, e cada um desses marcadores pode ter outros marcadores contidos neles...

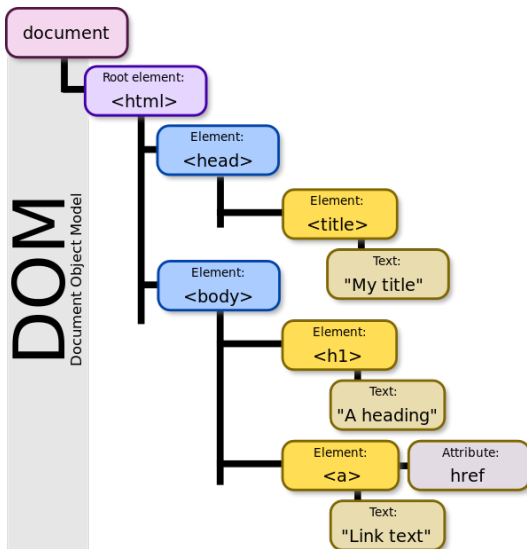
Exemplo 1

```
1 <html>
2   <head></head>
3   <body></body>
4 </html>
```

Exemplo 2

```
1 <table>
2   <tr>
3     <td></td>
4   </tr>
5 </table>
```

DOM - Document Object Model

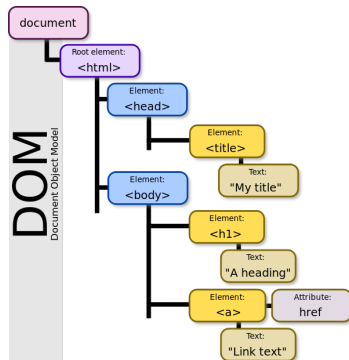


DOM - Document Object Model

- Como seria o documento HTML que gerou o DOM do slide anterior?

Exemplo

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>My title</title>
5   </head>
6   <body>
7     <a href='#'>Link text</a>
8     <h1>A heading</h1>
9   </body>
10 </html>
```



DOM - *Document Object Model*

- Porque estudar o DOM?
- O JS consegue recuperar a estrutura da página em um objeto:
 - ▷ Ou seja, temos acesso a toda estrutura da página em objetos no JS.
- Desta forma, podemos manipular páginas HTML utilizando JS. É possível, por exemplo:
 - ▷ Mudar o conteúdo de elementos HTML;
 - ▷ Mudar o estilo de elementos HTML;
 - ▷ Trabalhar com eventos;
 - ▷ Adicionar e remover elementos HTML.

Conteúdo

1 Introdução

2 Acessando elementos do HTML

DOM - *Document Object Model*

- Uma das grandes funcionalidade de JS é manipular páginas HTML, para deixá-las mais dinâmicas.
- A primeira coisa que precisamos fazer é encontrar um elemento do HTML que desejamos manipular. Para isso, JS oferece um método chamado `document.getElementById()`.
- O `document.getElementById()` permite referenciar dinamicamente qualquer elemento do documento HTML através do seu ID.
- Lembrete: um ID é um atributo definido nos marcadores:

Exemplo

```
1 <p id="par">Parágrafo com ID</p>
```

DOM - Document Object Model

Acessando um objeto pelo DOM

```
1  <!DOCTYPE html>
2  <html>
3  <head></head>
4  <body>
5      <p id="par">Parágrafo com ID</p>
6      <script>
7          const p = document.getElementById("par");
8          alert(p);
9      </script>
10 </body>
11 </html>
```

DOM - Document Object Model

Acessando um objeto pelo DOM

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <p id="par">Parágrafo com ID</p>
6   <script>
7     const p = document.getElementById("par");
8     alert(p);
9   </script>
10 </body>
11 </html>
```

- Teste o código acima. Em seguida, inverta a posição do script (coloque-o antes do parágrafo) e verifique a diferença na execução.

DOM - *Document Object Model*

- Note que no exemplo anterior o comando `alert()` não mostrou muita informação relevante:
▷ `[object HTMLParagraphElement]`
- Conseguimos entender que é um objeto do tipo parágrafo em HTML. Mas e o que exatamente há “dentro” dele?
- O JavaScript oferece um método para saber o que há dentro dos objetos JS:
▷ `console.log(obj);` - no **Firefox**
▷ `console.dir(obj);` - no **Chrome**
- Lembrando que para visualizar o resultado do método acima, devemos utilizar o menu de ferramentas de desenvolvedor. Nos navegadores Chrome ou no Firefox esse menu é exibido ao pressionar a tecla F12.

Acessando elementos pelo nome do marcador

- Existem outras formas de recuperar elementos HTML, não só pelo ID.
- É possível recuperar elementos pelo nome do marcador:
 - ▷ `const parag = document.getElementsByTagName("p");`
- Repare que o argumento deverá ser sempre o nome do marcador (Exemplos: `p`, `div`, `h1`, `table`, `ul`, *etc*).
 - ▷ Portanto, este método retorna **TODOS** os elementos com o nome do argumento.
 - ▷ No exemplo, ele retornará todos os parágrafos da página (na forma de um *array*).
 - ◇ Os elementos serão acessados como: `parag[0]`, `parag[1]`, ... e assim por diante.

Acessando elementos pelo nome da classe

- Também é possível recuperar elementos pelo nome da classe:
 - ▷ `const classes = document.getElementsByClassName("classe")`
- Este método retorna **TODOS** os elementos com o classe enviada no argumento:
 - ▷ No exemplo, retorna todos elemento que possuem o atributo: `class="classe"`;
- Como vários marcadores podem fazer parte de uma classe, o método também retorna um array de objetos.

Acessando elementos por seletores CSS

- É possível utilizar seletores CSS para acessar elementos do DOM:
 - ▷ `const objetos = document.querySelectorAll("p.importantes");`
- No exemplo acima, serão selecionadas todas as *tags* p (parágrafos) que estejam na classe importantes.
 - ▷ Ou seja, o resultado também é um array.
- Existe um outro método que retorna o primeiro objeto que corresponde ao seletor:
 - ▷ `const objeto = document.querySelector("p.importantes")`

Acessando elementos dentro de outros elementos

- O JavaScript permite recuperar os elementos de um elemento específico. Por exemplo, suponha que desejamos recuperar todos os parágrafos dentro da *tag* `main`:

Acessando parágrafos dentro do *main*

```
1 const obj_main = document.getElementById("main");  
2 const parágrafos = obj_main.getElementsByTagName("p");
```

Acessando os elementos dentro de um *form* com ID "form1"

```
1 const obj_form = document.forms["form1"];  
2 for (let i=0; i < obj_form.length; i++) {  
3   console.log(obj_form.elements[i]);  
4 }
```


Manipulando o HTML

- A manipulação de objetos HTML do JavaScript pode:

Escrever no documento HTML

```
1 document.write("<p>Mais um parágrafo</p>");
```

Mudar conteúdo HTML de um elemento

```
1 document.getElementById("id").innerHTML = "texto"  
2 // ou então:  
3 let variavel = document.getElementById("id")  
4 variavel.innerHTML = "texto"
```

Mudar o valor de um atributo

```
1 document.getElementById("id").src = "imagem.jpg"  
2 document.getElementById("tx-nome").value = "Rafael"  
3 // ou então:  
4 let variavel1 = document.getElementById("id")  
5 let variavel2 = document.getElementById("tx-nome")  
6 variavel1.src = "imagem.jpg"  
7 variavel2.value = "Rafael"
```

Manipulando o HTML

- Também é possível modificar estilos CSS:
 - ▷ Usando a propriedade: `.style.propriedade`
 - ▷ Ou também: `.style["propriedade"]`
- Exemplos:

Modificando a cor de fundo

```
1 document.getElementById("id").style.background = "#0F0";  
2 // ou então:  
3 document.getElementById("id").style["background"] = "#0F0";  
4 // ou então:  
5 let variavel = document.getElementById("id")  
6 variavel.style.background = "#0F0";
```

Modificando margens

```
1 document.getElementById("id").style.margin = "10px";
```

Modificando o display

```
1 document.getElementById("id").style.display = "none";
```

- E várias outras propriedades...

Adicionando novos elementos

- Podemos criar e adicionar novos elementos à elementos já existentes:

Código HTML

```
1 <div id="div1">
2 </div>
```

Código JS

```
1 let div = document.getElementById("div1");
2 let paragrafo = document.createElement("p"); // cria uma tag p (parágrafo)
3 div.appendChild(paragrafo); // adiciona o parágrafo criado como filho da div
```

- Rode o código acima e veja o HTML gerado pelo navegador.

Adicionando novos elementos

- Podemos criar e adicionar novos elementos à elementos já existentes:

Código HTML

```
1 <div id="div1">  
2 </div>
```

Código JS

```
1 let div = document.getElementById("div1");  
2 let paragrafo = document.createElement("p");  
3 paragrafo.innerHTML = "Texto do parágrafo";  
4 paragrafo.style.color = "magenta";  
5 div.appendChild(paragrafo);
```

- Rode o código acima e veja o HTML gerado pelo navegador.