Desenvolvimento Web

Pesquisa de DOM para JavaScript

Nome: Guilherme Max Pereira; Victor Pereira da Palma

Turma: 2TADSB

Curso: Análise e Desenvolvimento de Sistemas

DOM: SUA PÁGINA NO MUNDO JAVASCRIPT

Para permitir alterações na página, ao carregar o HTML da página, os navegadores carregam em memória uma estrutura de dados que representa cada uma das nossas tags no JavaScript. Essa estrutura é chamada de DOM (Document Object Model). Essa estrutura pode ser acessada através da variável global document .

O termo "documento" é frequentemente utilizado em referências à nossa página. No mundo front-end, documento e página são sinônimos.

QUERYSELECTOR

Antes de sair alterando nossa página, precisamos em primeiro lugar acessar no JavaScript o elemento que queremos alterar. Como exemplo, vamos alterar o conteúdo de um título da página. Para acessar ele:

document.querySelector("h1")

Esse comando usa os seletores CSS para encontrar os elementos na página. Usamos um seletor de nome de tag mas poderíamos ter usado outros:

document.querySelector(".class")

document.querySelector("#id")

ELEMENTO DA PÁGINA COMO VARIÁVEL

Se você vai utilizar várias vezes um mesmo elemento da página, é possível salvar o resultado de qualquer querySelector numa variável:

var titulo = document.querySelector("h1")

Executando no console, você vai perceber que o elemento correspondente é selecionado. Podemos então manipular seu conteúdo. Você pode ver o conteúdo textual dele com:

titulo.textContent

Essa propriedade, inclusive, pode receber valores e ser alterada:

titulo.textContent = "Novo título"

QUERYSELECTORALL

As vezes você precisa selecionar vários elementos na página. Várias tags com a classe .cartao por exemplo. Se o retorno esperado é mais de um elemento, usamos querySelectorAll que devolve uma lista de elementos (array).

document.querySelectorAll(".cartao")

Podemos então acessar elementos nessa lista através da posição dele (começando em zero) e usando o colchetes:

// primeiro cartão

document.querySelectorAll(".cartao")[0]

ALTERAÇÕES NO DOM

Ao alterarmos os elementos da página, o navegador sincroniza as mudanças e alteram a aplicação em tempo real.

FUNÇÕES E OS EVENTOS DO DOM

Apesar de ser interessante a possibilidade de alterar o documento todo por meio do JavaScript, é muito comum que as alterações sejam feitas quando o usuário executa alguma ação, como por exemplo, mudar o conteúdo de um botão ao clicar nele e não quando a página carrega. Porém, por padrão, qualquer código colocado no <script> , como fizemos anteriormente, é executado assim que o navegador lê ele.

Para guardarmos um código para ser executado em algum outro momento, por exemplo, quando o usuário clicar num botão, é necessário utilizar alguns recursos do JavaScript no navegador.

Primeiro vamos criar uma função:

function mostraAlerta() { alert("Funciona!"); }

Ao criarmos uma função, simplesmente guardamos o que estiver dentro da função, e esse código guardado só será executado quando chamarmos a função, como no seguinte exemplo: function mostraAlerta() {

alert("Funciona!");

}

// fazendo uma chamada para a função mostraAlerta, que será executada nesse momento

mostraAlerta()

Para chamar a função mostraAlerta só precisamos utilizar o nome da função e logo depois abrir e fechar parênteses. Agora, para que essa nossa função seja

chamada quando o usuário clicar no botão da nossa página, precisamos do seguinte código:

```
function mostraAlerta() { alert("Funciona!");

}

// obtendo um elemento através de um seletor de ID

var botao = document.querySelector("#botaoEnviar");

botao.onclick = mostraAlerta;
```

Note que primeiramente foi necessário selecionar o botão e depois definir no onclick que o que vai ser executado é a função mostraAlerta . Essa receita será sempre a mesma para qualquer código que tenha que ser executado após alguma ação do usuário em algum elemento. O que mudará sempre é qual elemento você está selecionando, a qual evento você está reagindo e qual função será executada.

Quais eventos existem

Existem diversos eventos que podem ser utilizados em diversos elementos para que a interação do usuário dispare alguma função:

oninput: quando um elemento input tem seu valor modificado

onclick: quando ocorre um click com o mouse

ondblclick: quando ocorre dois clicks com o mouse

onmousemove: quando mexe o mouse

onmousedown: quando aperta o botão do mouse

onmouseup: quando solta o botão do mouse (útil com os dois acima para gerenciar drag'n'drop)

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)

Existem também uma série de outros eventos mais avançados que permitem a criação de interações para drag-and-drop, e até mesmo a criação de eventos customizados.

FUNÇÕES ANÔNIMAS

No exercício anterior nós indicamos que a função mostraTamanho deveria ser executada no momento em que o usuário inserir o tamanho do produto no . Note que não estamos executando a função mostraTamanho, já que não colocamos os parênteses. Estamos apenas indicando o nome da função que deve ser executada.

```
inputTamanho.oninput = mostraTamanho

function mostraTamanho(){

outputTamanho.value = inputTamanho.value

}
```

Há algum outro lugar do código no qual precisamos chamar essa função? Não! Porém, é pra isso que damos um nome à uma função, para que seja possível usá-la em mais de um ponto do código.

É muito comum que algumas funções tenham uma única referência no código. É o nosso caso com a função mostraTamanho . Nesses casos, o JavaScript permite que criemos a função no lugar onde antes estávamos indicando seu nome.

```
inputTamanho.oninput = function() {

 outputTamanho.value = inputTamanho.value

}
```

Transformamos a função mostraTamanho em uma função sem nome, uma função anônima. Ela continua sendo executada normalmente quando o usuário alterar o valor para o tamanho.

MANIPULANDO STRINGS

Uma variável que armazena um string faz muito mais que isso! Ela permite, por exemplo, consultar o seu tamanho e realizar transformações em seu valor.

```
var empresa = "Caelum";

empresa.length; // tamanho da string

empresa.replace("lum","tano"); // retorna Caetano
```

A partir da variável empresa , usamos o operador ponto seguido da ação replace .

IMUTABILIDADE

String é imutável. Logo, no exemplo abaixo, se a variável empresa for impressa após a chamada da função replace , o valor continuará sendo "Caelum".

Para obter uma string modificada, é necessário receber o retorno de cada função que manipula a string, pois uma nova string modificada é retornada:

var empresa = "Caelum"; // substitui a parte "lum" por "tano"

empresa.replace("lum","tano");

console.log(empresa); // imprime Caelum, não mudou!

empresa = empresa.replace("lum","tano"); console.log(empresa); // imprime Caetano, mudou!

CONVERSÕES

O JavaScript possui funções de conversão de string para number:

var textoInteiro = "10";

var inteiro = parseInt(textoInteiro);

var textoFloat = "10.22";

var float = parseFloat(textoFloat);

MANIPULANDO NÚMEROS

Number, assim como string, também é imutável. O exemplo abaixo altera o número de casas decimais com a função toFixed . Esta função retorna uma string, mas, para ela funcionar corretamente, seu retorno precisa ser capturado:

var milNumber = 1000;

var milString = milNumber.toFixed(2); // recebe o retorno da função

console.log(milString); // imprime a string "1000.00"

CONCATENAÇÕES

É possível concatenar (juntar) tipos diferentes e o JavaScript se encarregará de realizar a conversão entre os tipos, podendo resultar em algo não esperado.

String com String

var s1 = "Caelum"; var s2 = "Inovação";

console.log(s1 + s2); // imprime CaelumInovação

```
var s1 = "Caelum";
 var s2 = "Inovação";
console.log(s1 + s2); // imprime CaelumInovação
```

String com outro tipo de dados

Como vimos, o JavaScript tentará ajudar realizando conversões quando tipos diferentes forem envolvidos numa operação, mas é necessário estarmos atentos na maneira como ele as realiza:

```
var num1 = 2;
var num2 = 3;
var nome = "Caelum"
// O que ele imprimirá?
// Exemplo 1:
console.log(num1 + nome + num2); // imprime 2Caelum3
// Exemplo 2:
console.log(num1 + num2 + nome); // imprime 5Caelum
// Exemplo 3:
console.log(nome + num1 + num2); // imprime Caelum23
// Exemplo 4:
 console.log(nome + (num1 + num2)); // imprime Caelum5

 // Exemplo 5:
console.log(nome + num1 * num2); // imprime Caelum6
// A multiplicação tem precedência
```

NaN

Veja o código abaixo:

```
console.log(10-"curso")
```

O resultado é `NaN` (not a number). Isto significa que todas operações matemáticas, exceto subtração, que serão vistas mais a frente, só podem ser feitas com números. O valor `NaN` ainda possui uma peculiaridade, definida em sua especificação:

```
var resultado = 10-"curso"; // retorna NaN
resultado == NaN; // false
NaN == NaN; // false
```

Não é possível comparar uma variável com `NaN`, nem mesmo `NaN` com `NaN`! Para saber se uma variável é `NaN`, deve ser usada a função **isNaN**:

```
var resultado = 10-"curso";
isNaN(resultado); // true
```

Referência Bibliográfica

Desenvolvimento Web com HTML, CSS e JavaScript (caelum.com.br)