Javascript

Do básico ao avançado seguindo a MDN web docs

João Vitor Rezende Moura - 1221184773

UNIT - Universidade Tiradentes

Abstract

abstract

Keywords: Javascript, Front-End, UI, media-queries, web

---

1 Primeiros passos com javascript

2 Criando elementos em javascript

2.1 Condicionais dentro do javascript

Dentro de qualquer linguagem de programação, temos que tomar decisões, sejam elas baseadas nas informações providas pelo usuário, nas informações que conseguimos captar dele, ou seja por algum outro motivo. Esse tipo de escolha, pode ser feito por meio de uma estrutura que existem em todas as linguagens de programação, e que recebe o nome de ***condicionais***.

Elas permitem que a partir de determinada condição que deve ser analisada, executarmos uma determinada ação, ou outra. Um exemplo de código javascript que possui essa determinada estrutura, é a seguinte:

```
if(condição){
  // codigo executado se a condição for verdadeira
}else{
  // código executado se a condição for falsa
}
```

A priori, temos que as palavras `if` e `else` são reservadas dentro do javascript para a construção de estruturas condicionais, portanto temos que dentro dos parênteses, temos a condição que deve ser analisada, e dentro das chaves, os blocos de códigos a serem executados condicionalmente.

A posteriori, temos que as vezes as comparações condicionais de dois fatores podem não ser suficientes para nós conseguirmos analisar o código e executá-lo da maneira desejada, portanto, temos a necessidade de comparar um determinado valor a várias possibilidades as quais ele pode assumir, e assim surge o `else if`:

```
if(primeiraCondicao){
  // executar código do primeiro bloco
}else if(segundaCondicao){
  // executar o segundo bloco de código
}else{
  // executar bloco de código quando condições não forem atingidas
}
```

2.2 Aninhando condicionais, e estabelecendo condições

Temos que as condições presentes analisandas dentro das condicionais, devem assumir valores booleanos, ou seja, de verdadeiro ou falso. Entretanto, podemos criar análises numéricas, decimais, e de outros tipos primitivos, que possuem como resultado, um valor booleano, e inserir como a condição que controla o fluxo da condicional, como no seguinte caso:

```
let condicao = 15 < 20

if(condicao){
  // executar código se a condicao for verdadeira
}else{
  // executar código se a condicao for falsa
```

Temos no exemplo anterior, que uma comparação lógica entre dois elementos numéricos gerou um valor booleano, e isso pode ser feito de várias maneiras por meio dos operadores lógicos, ou por meio de operadores de comparação. Os operadores que possibilitam isso, são os de comparação matemática, e operadores booleanos, que são o `&&` e o `||`, que são de disjunção e conjunção respectivamente

Podemos também aninhar condicionais, ou seja, dentro de alguma determinada condicional, fazer outra condicional que pode comparar o mesmo valor, ou um valor que não possua conexão com o primeiro comparado.

2.3 Instrução switch

As isntruções condicionais são muito úteis, entretanto, quando temos uma variável que pode assumir diversos valores, isso cria uma cadenia de condicionais que é dificil de manter no código, e difícil de entender, portanto, é recomendando que quando comparamos condicçoes que podem assumir diversos valores, que usemos a estrutura `switch`.

```
switch(expression){
  case choice1:
    // run code and break the comparation
    break;
  case choice2:
    // run code and break the comparation
    break;
  default:
    // only need to run the code actually
}
```

Nesse caso, temos que dois casos existem, e que quebramos a comparação como última instrução do caso o qual o mesmo corresponde. Isso se deve ao fato de que o switch case pode desejar que uma expressão, encaixe em mais de um caso, sendo assim, o mesmo deve passar pelos dois casos, e executar os dois blocos de isntrução, e isso é feito não quebrando a comparação dos casos, ou seja, não adicionando a instrução `break`. Caso nenhum dos casos seja verdadeiro, ou caso o que seja, não possua a instrução de quebrar a comparação, e seja o único verdadeiro, a expressão vai cair no caso padrão, e após executar as instruções, vai quebrar a comparação por ser o último caso propositadamente.

2.4 Operador ternário

Temos que as comparações são elementos essenciais dentro da programação, entretanto elas ocupam muito espaço visual, e algumas comparações são tão pequenas que são quase "banais", pra isso existe o operador ternário, que permite uma comparação de um elemento, que pode receber um de dois valores se sua condição for verdadeira, e essa instrução é feita toda em um única linha.

```
(condição) ? se a condição for verdadeira : se a condição for falsa;
```

3 Loops

Os loops. são estruturas que permitem repetição de determinadas partes do código, até que uma determinada condição que pode ser pré-definida, ou mudar conforme a execução do código, seja realizada. Eles possuem uma estrutura padrão, mas podem se modificar conforme as necessidades e peculiaridades de cada código. Por padrão os laços dentro do Javascript são feitos da seguinte forma:

```
for(inicializador; condição de saída; expressão final){
  //código
}
```

3.1 Saindo dos laços por meio da instrução break

Apesar dos laços possuirem uma quantidade definida de vezes as quais eles repetem um conjunto específico de ações, eles podem sofrer alterações no seu comportamente e podem também sofrerem desvios por meio de condicionais que podem ser colocadas dentro dos laços, com intuito de caso uma certa condição seja apresentada, aquela interação e as seguintes sejam canceladas.

```
for(inicializador; condição de saida; expressao final){

  // codigo
  if(condicao){
    break;
  }
}
```

Obs: O mesmo pode ser feito com a instrução **continue;**, so que diferentemente, ele não vai cacelar todas as interações, ele só vai cancelar a atual, e prosseguir com as seguintes.

3.2 While e do...while

Essas instruções são parecidas com os laços, entretanto, a diferença é que elas apresentam em si, o fato de que ela não necessariamente possuem dentro de si uma condição de saída, mas sim uma condição de continuidade, em que caso ela seja satisfeita, aquele laço será repetido até que essa mesma condição seja quebrada.

Temos que essa instrução pode ser apresentada e executada de duas maneiras, na primeira, ele confere a condicional, e se for verdadeira, ele executa um bloco de código e parte para a próxima verificação. Por outro lado, podemos fazer com que ele primeiro execute a ação, e somente posteriormente verifique as condicionais, como mostrados os exemplos seguintes:

```
while(condicao){

  // executar código

}

do{
  \\ executar código

}while(condição)
```

4 Funções - Blocos reutilizáveis de código

Funções são rotinas de código pré-definidas que podem ser utilizadas diversas vezes, e que são agrupadas dentro de uma mesma chamada, ou seja, invocando somente um comando dentro do código, todas aquelas ações correspondentes presentes dentro da nossa função, será executadas.

Elas podem vir de natureza da própria linguagem, serem chamadas a partir de objetos, ou serem criadas pelo desenvolvedor para executar uma rotina específica de funções que existem, elas podem aparecer das seguintes formas:

```
// embutida do navegador
console.log();
// por meio de objetos
MyArray.join("");
// funções criadas pelo desenvolvedor
function draw(){
  console.log("desenhando");
}
draw();
```

4.1 Funções vs. Métodos

Uma coisa técnicas mas que pode ser pontuada, é o fato de que as funções embutidas nos navegadores na verdade são chamadas de métodos, pois o navegador é um objeto, e toda função que referencia um objeto é chamada de método, e as funções não necessariamente referenciam objetos.

As funções dentro do javascript, podem possuir uma característica muito especial, que é serem anônimas, ou seja, não possuírem um nome definido que as permitam serem invocadas. Normalmente fazemos isso quando criamos uma função junto com um manipulador de eventos.

4.2 Parâmetros de funções

Algumas funções requerem parâmetros a serem especificados quando estamos invocando eles, esses valores precisam ser inclusos dentro dos parênteses da função, o que é necessário para fazer seu trabalho apropriado.

4.3 Função escopo e conflitos

Um conceito muito importante quando lidamos com funções é o escopo do seu uso. As funções, e suas variáveis possui um escopo, ou seja eles funcionam apenas dentro do seu próprio funcionamento, sendo inacessível para outras funções ou de código do lado de fora das funções.

Essa "passagem" de valores só pode ser feito por meio da passagem dos valores do escopo local, por meio ad chamada da outra função, e a enviando-a como parâmetro.

4.4 Retornando valores

As funções elas podem ser entendidas, como rotinas de ações que executam e modificam dados, portanto, pode-se desejar obter de volta os dados modificados, ou obter os dados que foram criados. Para isso, podemos usar a palavra reservada **return** para indicar um dado que deve ser retornado da chamada da função.

5 Introdução a eventos.

Eventos são ações ou ocorrências que acontecem no sistema que estamos desenvolvendo, no qual este te alerta sobre ações para que as quais devemos poder responder de alguma determinada forma. Eventos web são disparados dentro da janela do navegador e tendem a estarem anexados a lagum item específico nele.

Os **Manipuladores de eventos** são blocos de código que são executados pelo evento quando ele é disparado, esse elemento também pode ser chamado de *event handler*. Um exemplo de seu uso pode ser visto a seguir:

```
HTML
<button>Change color</button>


JS
var button = document.querySelector("button")
```

```
function random(number){
  return Math.floor(Math.random() * (number + 1));
}

button.onClick = function(){
  var randomColor = "rgb(" + random(255) +
                       "," + random(255) +
                       "," + random(255) + ")"
  document.body.style.backgroundColor = randomColor;
```

5.1 Não são apenas páginas web

O modelo de eventos no JavaScript para web pages difere dos outros modelos de eventos próprios que as outras linguagens de programação podem conter. O **Node.js** é um interpretador de código JavaScript que permite o desenvolvimento de aplicações do lado do servidor. O event model do Node.js depende dos ouvintes para escutar eventos e emissores para emitir eventos periodicamente.

5.2 Maneiras de usar os eventos Web

Hpa várias maneiras de adicionar código com event listeners a uma determinada página web, mas a forma mais comum disso ser feita é por meio de propriedades nas páginas as quais são adicionadas os manipuladores de eventos. As **propriedades** podem ser usadas para conter código que manipulam os eventos. Essas propriedades são essencialmente iguais a qualquer outra disponível, mas com um tipo especial o qual pertencem.

5.3 addEventListener e removeEventListener

Temos que os mecanismos de eventos são definidos nas especificações de eventos que ficam presentes dentro da DOM(Document Object Model), o qual fornece aos navegadores uma nova função, que é a de adição de eventlisteners. Dentro dessas funções, especificamos o nome do evento para o qual queremos registrar esse manipulador, e o código que compreende a função do manipulador que queremos executar em resposta a ele.

```
let button = document.querySelector("button");

button.addEventListener("click", => (){
  document.body.style.backgroundColor = randomcolor...;
});


// Se quisermos remover o event listener, podemos usar o seguitne código

button.removeEventListener("click", colorChange)
```

Uma observação que deve ser feita sore o event listener e sua remoção, é que para a mesma seja feita, precisamos especificar o nome da função responsável pelo event listener.

5.4 Objetos de Eventos

Dentro de uma função a qual manipulaos o evento podemos desejar verificar um parâmetro chamado `event`. É um objeto representante do evento em si, e é passado automaticamente para os manipuladores de eventos para fornecer recursos e informações extras. Esse evento possui o atributo target, que é sempre uma referência ao elemento em que o evento acabou de ocorrer. Ele é útil quando desejamos definir o mesmo manipulador de eventos em vários elementos e fazer algo com todos eles quando ocorre um evento neles. Isso pode ser feito com o método `querySelectorAll()`.

Outra coia importante é que podemos implementar uma verificação simples dentro do manipulador de eventos, como dentro do `onsubmit`, por exemplo, que verifica se a função no objeto do evento tem campos vazios.

```
var form = document.querySelector("form");
var fname = document.getElementById("fname");
var lname = document.getElementById("lname");
var submit = document.getElementById("submit");
var para = document.querySelector("p");

form.onsubmit = function (e) {
  if (fname.value === "" || lname.value === "") {
    e.preventDefault();
    para.textContent = "You need to fill in both names!";
  }
};
```

5.5 Borbulhamento e Captura de eventos

Borbulhamento e captura de eventos são dois mecanismos que descrevem oque acontece quando dos manipuladores do mesmo tipo de evento são ativados em um elemento. Quando temos dois manipuladores de eventos, quando um evento é acionado em um elemento que possui elementos pai, os navegadores executam duas fases diferentes, que são as fases de captura de bubbling.

5.5.1 Fase de Captura

Na fase de captura, o nabegador verifica se o ancestral mais externo do elemento tem um manipulador de eventos registrado nele na fase de captura e executa em caso afirmativo, o manipulador verificado é o `onclick`. Em seguida ele passa para o próximo elemento dentro do html e faz a mesma coisa, e assim por diante até alcançar o elemento que foi realmente clicado.

5.5.2 Fase de bubbling

O navegador verifica se o elemento que foi clicado realmente tem o manipulador de eventos específico registrado nele e o executa em caso afirmativo. Em seguida ele passa para o próximo elemento até alcançar o elemento html

Esse tipo de problema pode ser corrigido com o objeto padrão chamado o método `stopPropagation()`, que quando invocado no objeto de evento de um manipulador faz com que o manipulador seja executado, mas o evento não borbulha mais acima da cadeia.

6 Intodução a Objetos em JavaScript

6.1 Noções básicas de Objetos

Um objeto é uma coleção de dados e/ou funcionalidades relacionadas. Eles podem ser armazenados dentro de variáveis, como valores descritos, para criar um objeto vazio, podemos fazer da seguinte maneira:

```
var pessoa = {};

// Console
[object Object]
```

Entretanto, podemos criar objetos pré-definidos para o uso, possuindo atributos e métodos já instanciados dentro da sua criação, como mostrado a seguir:

```
verbatim
```