

Eventos no DOM

Davi Valadares Rodrigues Feliciano

Driven Education

03 de fevereiro de 2023

O que são eventos?

No contexto geral

- É uma ocorrência reconhecível por *software*, um sinal emitido que pode ser identificado e interpretado
- Normalmente originada assincronamente por uma fonte externa
- Após interpretar um evento, um *software* pode lidar de acordo com o tipo do evento em sua execução

Exemplos

- Exemplos de fontes são o usuário por meio de periféricos ou mesmo outros *softwares*
- O próprio modelo de *runtime* do JavaScript consiste em um loop de eventos

O que são eventos?

No contexto do DOM

- Eventos são disparados frequentemente ao longo do uso de uma página o *browser*
- Esses eventos normalmente estão acoplados a um ou múltiplos elementos

Exemplos de Eventos

- Clicar em um elemento
- Pressionar teclas
- *Drag and drop*
- *Scroll*
- Copiar, cortar e colar
- *Reset ou submit* de *forms*
- No carregamento da página
- Redimensionar a janela

O que são eventos?

No contexto do DOM

- Para reagir à ocorrência desses eventos, devemos capturá-los de alguma forma
- Eventos não capturados disparam comportamentos padrões
- Para isso usamos *event listeners* e *event handlers*
- *Event listeners*, como o próprio nome diz, quando ativados, aguardam o disparo de um determinado evento
- Quando este ocorre, o *event handler*, que é uma função previamente definida é executado

Criando *event listeners* no JavaScript

- Para criar um *event listener* usamos o método `addEventListener` de objetos do tipo `Element`
- Passamos como primeiro parâmetro uma string identificadora do tipo de evento a ser capturado
- Como segundo argumento, passamos uma função de *callback* que será executada em cada ocorrência do evento
- Para essa função de *callback* é passado uma instância de um objeto `Event`

O que é uma função de *callback*?

- É uma função passada como argumento para outra função
- Será chamada em algum momento na execução da última

Objetos Event

```
function eventHandler(event) {  
    console.log(event.type); // "click"  
    console.log(event.timeStamp); // 15965.7999999702  
    console.log(this === event.currentTarget); // true  
}  
  
element.addEventListener("click", eventHandler);
```

Event.type	Uma string identificando o tipo do evento
Event.timeStamp	O momento em que o evento ocorreu, em milissegundos, em relação ao carregamento da página
Event.currentTarget	Uma referência ao elemento ao qual o listener foi acoplado
Event.target	Uma referência ao elemento no qual o evento teve origem (<i>event bubbling</i>)

Objetos Event

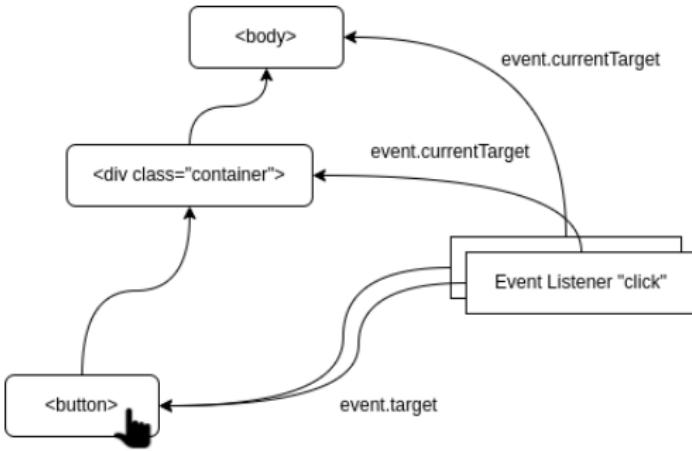
```
// Chame selectItem quando um card é clicado
const items = document.querySelectorAll(".item");
items.forEach((item) => {
  item.addEventListener("click", selectItem);
});

/* Coloque o foco no input #address ao pressionar
   Enter quando o input #fullname estiver em foco */
fullnameInput.addEventListener("keyup", (event) => {
  if (event.key === "Enter") {
    addressInput.focus();
  }
});

/* Chame showOrderModal ao pressionar Enter
   quando o input #address estiver em foco */
addressInput.addEventListener("keyup", (event) => {
  if (event.key === "Enter") {
    showOrderModal();
  }
});
```

Event Bubbling

- Quando um evento é criado, ele se propaga pelos níveis superiores da árvore de elementos do DOM, até chegar ao elemento raiz
- Isso significa que mesmo um elemento não tendo *event listeners* acoplados ainda pode disparar *event handlers* em seus pais



Event Bubbling

- Como prevenir esse comportamento quando indesejado? Com uma chamada ao método event.stopPropagation

```
<body>
  <div class="container">
    <button>Botão</button>
  </div>
</body>
```

```
const body = document.querySelector("body");
const container = document.querySelector(".container");

body.addEventListener("click", bodyEventHandler);

container.addEventListener("click", (event) => {
  event.stopPropagation();
  containerEventHandler(event);
});
```

addEventListener vs. onclick

- É possível adicionar múltiplos *event listeners* a um único elemento
- Como citado, existem vários tipos de evento que não são acessíveis em atributos do HTML
- *Event listener* possibilita o acesso a propriedades do evento por meio de objetos do tipo Event, o que pode ser útil em diversas aplicações