Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end 1 (DFE_1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Eventos

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Pelotas

Eventos

Um evento, no contexto de JavaScript e desenvolvimento web, é qualquer ação ou ocorrência que pode ser detectada pelo navegador.

São ações que o usuário pode realizar em elementos (**interações**). Por exemplo:

- Ações do usuário (cliques, movimentos do mouse, pressionar uma tecla);
- Mensagens de outras aplicações (uma resposta de uma chamada HTTP);
- Ou mesmo pelo próprio navegador (um página terminando de carregar).

Basicamente, **eventos são o modo pelo qual os usuários podem interagir com a página web**, permitindo que o código JavaScript responda a essas ações de maneira **dinâmica e interativa.**

Exemplos de eventos

onBlur	remove o foco do elemento
onChange	muda o valor do elemento
onClick	o elemento é clicado pelo usuário
onFocus	o elemento é focado
onKeyPress	o usuário pressiona uma tecla sobre o elemento
onLoad	carrega o elemento por completo
onMouseOver	define ação quando o usuário passa o mouse sobre o elemento
onMouseOut	define ação quando o usuário retira o mouse sobre o elemento
onSubmit	define ação ao enviar um formulário

Eventos

Qualquer ação que possa ser executada sobre elementos em uma página HTML.

Evento

Ações em elementos HTML

- Carregamento da página;
- Clique em um botão;
- Campo de formulário modificado;
- ...

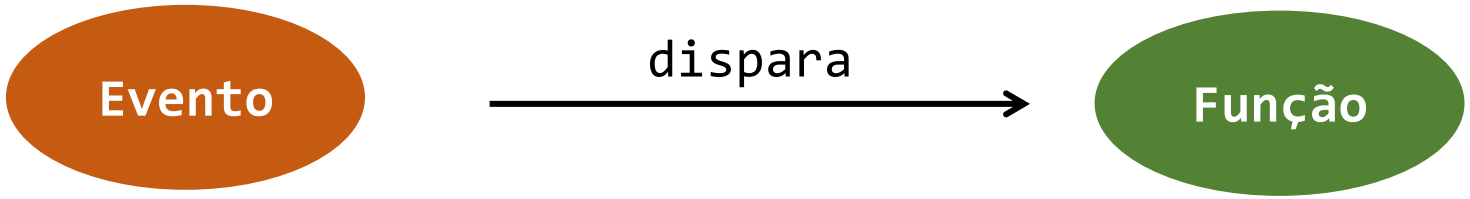
Mas o que acontece quando um evento ocorre?

W3C HTML DOM Events

MDN Event reference

Eventos x Funções

Trechos de códigos podem ser disparados em resposta a ocorrência de um evento!



Ações em elementos HTML

- Carregamento da página;
- Clique em um botão;
- Campo de formulário modificado;
- ...

Bloco de código executado quando é “chamado”

- Abrir uma janela de alerta;
- Modificar o árvore do documento HTML;
- Modificar o estilo de um elemento HTML;
- ....

Como aplicar eventos a elementos HTML?

Listeners (ouvintes)

Listeners, são **funções designadas para esperar e responder a certos eventos** que ocorrem dentro de uma aplicação ou sistema.

No contexto de desenvolvimento web, *listeners* são frequentemente associados a eventos no DOM, tais como:

- cliques do mouse;
- pressionamentos de teclas;
- alterações de formulário;
- carregamentos de página;
- etc...

Ao definir um *listener* para um evento específico em um elemento HTML, está instruindo o navegador a ficar "ouvindo" um evento específico.

Se e quando o evento ocorre, o navegador executa a função associada ao *listener*, permitindo uma interatividade dinâmica nas páginas web com base nas ações do usuário ou em outros eventos.

Existem algumas maneiras de configurar esses *listeners* em elementos HTML usando JavaScript...

1) Atributos de evento HTML

Eventos *inline*, ou seja tudo é definido **diretamente na tag do elemento**. Por exemplo:

```
  <body>
    <h1 onclick="this.innerHTML='Isso acontece ao usar o evento onClick!'">
      Clique no título para testar o evento onClick!
    </h1>

    <br>

    <button onclick="alert('Clicou!')">Clique-me</button>

  </body>
</html>
```

Essa abordagem é simples e direta, **mas mistura o conteúdo HTML com o código JavaScript**, o que pode complicar a manutenção do código e violar os princípios de separação de preocupações.

2) Propriedades de evento do JavaScript

No JavaScript, cada elemento DOM tem propriedades correspondentes a possíveis eventos.

É possível **atribuir uma função a essas propriedades diretamente no código JavaScript**.

```
<body>
  <button id="meuBotao"">Clique-me</button>

  <script>
    document.getElementById("meuBotao").onclick = function () {
      alert("Clicou!");
    };
  </script>
</body>
```

Esse método é **um pouco mais organizado** do que usar atributos de evento diretamente no HTML, pois mantém o código **JavaScript separado da marcação HTML**.

No entanto, **permite apenas um *listener*** por tipo de evento em cada elemento.

3) addEventListener()

Maneira **mais flexível e recomendada de adicionar *listeners*** aos elementos.

- Permite **adicionar múltiplos *listeners*** a um único elemento;
- Oferece **mais controle** sobre a captura e a propagação de eventos;
- É a **abordagem padrão** em aplicações modernas.

```
<body>
  <button id="meuBotao"">Clique</button>

  <script>
    document.getElementById("meuBotao")
      .addEventListener("click", function () {
        alert("Clicou com event listener!");
      });
  </script>
</body>
```

3) addEventListener()

Exemplo com **múltiplos manipuladores de eventos**.

```
<body>

  <button id="meuBotao"">Clique</button>

  <script>
  document.getElementById("meuBotao").addEventListener("click", function() {
      alert("Primeiro handler!");
  });

  document.getElementById("meuBotao").addEventListener("click", function() {
      alert("Segundo handler!");
  });

  </script>
</body>
```

4) removeEventListener()

Complementar ao `addEventListener()`, o `removeEventListener()` permite a remoção de listeners, algo especialmente útil em aplicações dinâmicas onde elementos são frequentemente adicionados, removidos ou têm seus *listeners* alterados.

Para remover um *event listener*, é necessário passar os mesmos argumentos usados para adicioná-lo com `addEventListener()`.

Isso inclui o tipo de evento, a função de callback e, se utilizado, o objeto de opções ou o booleano indicando se o evento é capturado ou não.

```
function cliqueBotao() {
          console.log('Botão clicado!');
      }

document.getElementById('meuBotao').addEventListener('click', cliqueBotao);


document.getElementById('meuBotao').removeEventListener('click', cliqueBotao);
```

Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end 1 (DFE_1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Eventos

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Pelotas