JavaScript: Fundamentos para Desenvolvimento Web Interativo - Turma 2025B

5.2 Formas de tratamento de eventos

O tratamento de eventos em JavaScript envolve especificar o que acontecerá quando o usuário realizar uma ação específica. Isso é geralmente feito definindo funções de manipulação de eventos e associando-as aos elementos do DOM que devem responder a esses eventos. Neste documento, estão algumas formas comuns de tratar eventos da web.

Definindo funções de manipulação de eventos

Uma função de manipulação de evento é uma função em JavaScript que contém o código que deve ser executado em resposta a um evento específico. Veja um exemplo no quadro abaixo:

```
function handleClick() {
      alert('O botão foi clicado!');
}
```

OBS: Os termos 'manipulador de eventos' (**event handler**) e 'ouvinte de eventos' (**event listener**) são frequentemente usados de forma intercambiável, mas tecnicamente, eles desempenham papéis complementares na gestão de eventos. O ouvinte de eventos é responsável por detectar a ocorrência de um evento e acionar a execução do manipulador de eventos. Por sua vez, o manipulador de eventos é o bloco de código que responde ao evento.

Associando manipuladores de eventos

Associar manipuladores de eventos em JavaScript é uma prática comum para adicionar interatividade a elementos HTML. Existem várias maneiras de associar esses manipuladores a elementos HTML. Veremos algumas delas no decorrer deste texto.

Atributos de evento no HTML

Anexar eventos diretamente a elementos HTML é uma técnica simples e direta de adicionar comportamentos interativos a elementos específicos em uma página da web. Isso é feito atribuindo valores a atributos de evento HTML, como **onclick**, **onmouseover**, **onkeydown**, entre outros.

Para anexar um evento diretamente a um elemento HTML, você inclui um atributo de evento no elemento desejado. O valor desse atributo é uma string de JavaScript contendo o código a ser executado quando o evento ocorre. Por exemplo, para anexar um evento de clique a um botão, você usaria o atributo **onclick**, seguido pelo código JavaScript que deseja executar quando o botão é clicado:

```
<button onclick="handleClick()">Clique em mim!</button>
```

Quando o evento ocorre (por exemplo, quando o botão é clicado), o navegador executa o código JavaScript especificado no valor do atributo de evento. No exemplo acima, uma caixa de diálogo de alerta será exibida quando o botão for clicado.

OBS: Este método é rápido e simples, mas mistura o código JavaScript com o HTML, o que não é recomendado para manutenção e escalabilidade.

Propriedades de evento no DOM

Uma abordagem mais poderosa e flexível é adicionar manipuladores de eventos usando JavaScript. Isso mantém o seu HTML limpo e o JavaScript separado, facilitando a manutenção. Por exemplo, no HTML teremos o seguinte código:

```
<button id="meuBotao">Clique em mim!</button>
```

E no JavaScript teremos o seguinte código:

```
function handleClick() {
    alert('O botão foi clicado!');
}

const botao = document.getElementById('meuBotao');
botao.onclick = handleClick;
```

Método addEventListener

O método **addEventListener** permite aos desenvolvedores especificar um ouvinte de eventos (**event listener**) que reage a eventos específicos, como cliques, movimentos de mouse, alterações de formulário, entre outros. Ele fornece uma maneira flexível e poderosa de lidar com eventos, permitindo que você adicione e remova manipuladores de eventos de forma dinâmica.

Veja no quadro abaixo sua sintaxe:

```
element.addEventListener(event, function, useCapture);
```

Onde:

- **element:** O elemento DOM ao qual o ouvinte de evento será adicionado.
- **event:** Uma string representando o tipo de evento a ser escutado (por exemplo, '**click**', '**mouseover**', '**keydown**', etc.).
- **function:** A função que será chamada quando o evento ocorrer. Esta função é frequentemente chamada de "**callback**".
- **useCapture (opcional):** Um booleano que indica se o evento deve ser capturado durante a fase de captura ou manipulado durante a fase de borbulhamento. O valor padrão é false, o que significa que o evento será manipulado na fase de borbulhamento.

Neste exemplo abaixo, um ouvinte de evento é adicionado a um botão para alertar o usuário cada vez que o botão for clicado.

```
function handleClick() {
      alert('O botão foi clicado!');
}

const botao = document.getElementById('meuBotao');
botao.addEventListener('click', handleClick);
```

Veja que ao utilizar o método **addEventListener**, estamos definindo um ouvinte de eventos que 'escuta' um tipo específico de evento, neste caso um clique. Quando esse evento é detectado, o código associado — o manipulador de eventos — é executado para responder a esse evento.

OBS: Quando um evento é disparado em um elemento que está dentro de uma hierarquia de elementos (como um elemento **<button>** dentro de outros contêineres HTML, tal como uma ** **), os navegadores processam o evento em duas fases distintas: a fase de captura e a fase de borbulhamento.

- **Captura:** é a primeira fase na propagação de eventos. Durante esta fase, o evento é capturado inicialmente pelo elemento raiz do documento (**<html>**) e propaga-se para baixo, através do DOM, até o elemento onde o evento foi realmente disparado (o elemento de destino). O propósito da fase de captura é oferecer uma maneira de interceptar eventos antes que eles cheguem ao seu destino.
- **Borbulhamento:** após o evento ser disparado no elemento de destino, ele entra na fase de borbulhamento, que é o processo inverso da captura. Durante o borbulhamento, o evento viaja de volta ao elemento raiz do documento (**<html>**), passando por cada elemento pai sucessivamente até que chegue ao topo.Logo, quando um evento ocorre em um elemento, como um clique em um botão, o evento não é limitado apenas a esse elemento; em vez disso, ele se propaga ou "borbulha" através da hierarquia dos elementos pai até o elemento raiz do documento. Por exemplo, se você tem um botão dentro de um **div** dentro de uma seção dentro do corpo do documento, e você clica no botão, o evento de clique será acionado primeiro no botão, depois no **div**, depois na seção e, finalmente, no corpo do documento.

Passando eventos para manipuladores

Quando você está trabalhando com eventos em JavaScript, frequentemente você precisará acessar detalhes específicos sobre o evento que foi disparado. Isso é feito passando um objeto de evento (**Event**) para a função de manipulação de eventos (**event handler**). Este objeto de evento contém uma série de informações úteis sobre o evento, como o tipo de evento, o alvo do evento, a posição do mouse (para eventos de mouse) e muito mais.

Quando você define um manipulador de evento usando **addEventListener**, o navegador automaticamente passa um objeto de evento como o primeiro argumento para a função de manipulação sempre que o evento ocorre. Você não precisa fazer nada especial para receber este objeto; você simplesmente o define como um parâmetro na função de manipulação. Veja um exemplo no quadro abaixo:

```
element.addEventListener('click', function (event) {
      console.log(event);
});
```

Neste exemplo, **event** é um objeto que contém informações sobre o evento de clique. Este objeto de evento tem muitas propriedades úteis, tais como:

- **type:** Indica o tipo de evento (por exemplo, '**click**', '**keydown**').
- **target:** O elemento que foi o alvo real do evento.
- **currentTarget:** O elemento ao qual o manipulador de eventos foi realmente anexado (útil durante a captura e o borbulhamento).
- **preventDefault():** Um método que pode ser chamado para prevenir a ação padrão que pertence ao evento (por exemplo, evitar que um formulário seja enviado).
- **stopPropagation():** Um método usado para parar a propagação do evento para os próximos ouvintes de eventos na fase de captura ou borbulhamento.
- **keyCode (para eventos de teclado):** O código da tecla que foi pressionada.
- **clientX e clientY (para eventos de mouse):** As coordenadas do mouse quando o evento ocorreu.

Veja o exemplo no quadro abaixo:

```
function handleClick(event) {
  console.log('Tipo de evento:', event.type);
  console.log('Elemento alvo:', event.target);
  console.log('Elemento que registrou o manipulador:', event.currentTarget);
  event.preventDefault(); // Impede qualquer ação padrão do botão, se houver
}

const botao = document.getElementById('meuBotao');
botao.addEventListener('click', handleClick);
```

Veja o resultado no navegador quando clicamos no botão:

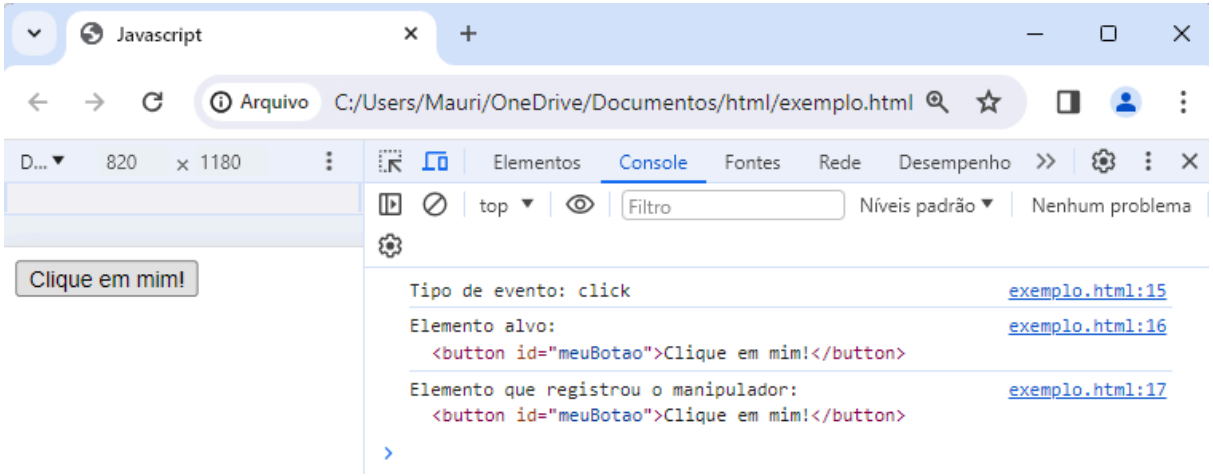

Figura 1 - Exemplo de como passar um objeto de evento (Event) para a função de manipulação de eventos (event handler)

Descrição da imagem: Na página, há um botão com o texto "Clique em mim!" e o id "meuBotao". Quando o botão é clicado, a função handleClick é executada, realizando as seguintes ações: 1) e xibe no console do desenvolvedor do navegador o tipo de evento, que será "click"; 2) exibe no console o elemento alvo do evento, que é o botão clicado; 3) exibe no console o elemento que registrou o manipulador, que também é o botão clicado; 4) i mpede qualquer ação padrão do botão usando event.preventDefault(), caso exista alguma.

Removendo manipuladores de eventos

Você também pode remover um manipulador de eventos se ele não for mais necessário, usando o método **removeEventListener**. É importante notar que você deve passar exatamente a mesma função de manipulação de evento que foi passada ao **addEventListener**.

Veja um exemplo no quadro abaixo:

```
function handleClick() {
      alert('O botão foi clicado!');
}

const botao = document.getElementById('meuBotao');

// Adicionando o evento
botao.addEventListener('click', handleClick);

// Removendo o evento
botao.removeEventListener('click', handleClick);
```

◄ 5.1 Introdução ao eventos do navegador

Seguir para...

5.3 Tipos de eventos ►