

JavaScript Eventos

5.1 O que são eventos?

Os eventos são a maneira que temos em Javascript de **controlar as ações** dos usuários e definir um comportamento da página quando os eventos ocorrem. Quando um usuário visita uma página web e **interage** com ela, produzem-se eventos e com Javascript podemos definir o que queremos que ocorra quando os eventos ocorrem. Por exemplo, podemos definir o que acontece quando o usuário **clica sobre um botão, edita um campo de texto ou abandona a página**.

5.2 Principais eventos implementados pelo JavaScript

Evento	Descrição
Abort (onAbort)	Este evento acontece quando o usuário interrompe o carregamento de um recurso
Blur (onBlur)	Ocorre quando o elemento perde o foco , ou seja, quando o usuário clica fora do elemento, este não será mais selecionado como estando ativo.
Change (onChange)	Ocorre quando o usuário altera o conteúdo de um campo de dados de um formulário.
Click (onClick)	Ocorre quando o usuário clica no elemento associado ao evento.
dblclick (onDblclick)	Ocorre quando o usuário clica duas vezes no elemento associado ao evento (um hiperlink ou um elemento de formulário). Este evento só é suportado pelas versões do JavaScript 1.2 e superior
dragdrop (onDragdrop)	Ocorre quando o usuário efetua um <i>arrastar-largar</i> na janela do navegador. Este evento só é suportado pelas versões do JavaScript 1.2 e superior
error (onError)	É desencadeada quando ocorre um erro durante o carregamento da página. Este evento faz parte do JavaScript 1.1.
Focus (onFocus)	Ocorre quando o usuário dá foco a um elemento, isso significa que este elemento foi selecionado como estando ativo
keydown (onKeyDown)	Ocorre quando o usuário pressiona uma tecla do seu teclado. Este evento só é suportado pelas versões do JavaScript 1.2 e superior
keypress (onKeyPress)	Ocorre quando o usuário mantém pressionada uma tecla do seu teclado. Este evento só é suportado pelas versões do JavaScript 1.2 e superior
keyup (onKeyPress)	Ocorre quando o usuário solta uma tecla do seu teclado antpressionada anteriormente. Este evento só é suportado pelas versões do JavaScript 1.2 e superior
Load (onLoad)	Ocorre quando o navegador do usuário carrega a página em curso
MouseOver (onMouseOver)	Ocorre quando o usuário põe o cursor do mouse acima de um elemento
MouseOut (onMouseOut)	Ocorre quando o cursor do mouse deixa um elemento. Este evento faz parte do Javascript 1.1.
MouseMove	Quando o cursor do mouse é movido

Reset (onReset)	Ocorre quando o usuário apaga os dados de um formulário com o botão Reset.
Resize (onResize)	Ocorre quando o usuário redimensiona a janela do navegador
Select (onSelect)	Ocorre quando o usuário seleciona um texto (ou uma parte de um texto) em um campo do tipo "text" ou "textarea"
Submit (onSubmit)	Ocorre quando o usuário clica no botão de submissão de um formulário (o botão que permite enviar o formulário)
Unload (onUnload ou onBeforeUnload)	Ocorre quando o navegador do usuário sai da página em curso

5.3 Gestões de eventos disponíveis em JavaScript

Cada elemento da página tem sua própria lista de eventos suportados.

OBJETO	GESTÃO DE EVENTO DISPONÍVEIS
Janela	onload, onlnload.
Link hipertexto	onclick, onmouseover, on mouseout.
Elemento de texto	onblur, onchange, onfocus, onselect.
Elemento de área de texto	onblur, onchange, onfocus, onselect.
Elemento botão	onclick.
Botão Radio	onclick
Lista de seleção	onblur, onchange, onfocus
Botão Submit	onclick
Botão Reset	onclick

5.4 Definição de Manipuladores de Eventos

Em JavaScript pode-se definir a manipulação de eventos utilizando diversas estratégias.

É possível inserir JavaScript **diretamente dentro do atributo**, como no exemplo abaixo, porém, **isso é uma prática ruim**. Pode parecer fácil usar um atributo manipulador de eventos se você estiver apenas fazendo algo realmente rápido, mas eles se tornam rapidamente incontroláveis e ineficientes.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Teste de Eventos</title>
</head>
<body id='bd1'>
  <h1> Teste de eventos </h1>
```

```

        <input type="button" id="btnteste" name="btntestel" value="Teste"
onclick="alert('Olá, esta é a forma mais antiga de manipulação de eventos');" />
        <p id='msg'></p>
</body>
</html>

```

Para começar, não é uma boa ideia misturar o código HTML e o JavaScript, pois é difícil analisar/manter. Manter o código JavaScript em um só lugar é melhor; se estiver em um arquivo separado, você poderá aplicá-lo a vários documentos HTML.

Outra maneira de utilizar eventos consiste em **chamar a execução de uma função diretamente no código HTML**, conforme o exemplo abaixo (esta função pode estar em um arquivo javascript externo)

HTML

```

<input type="button" id="btnteste" name="btntestel" value="Teste"
onclick=bgChange () />

```

JavaScript

```

function bgChange() {
    var rndCol = 'rgb(' + Math.random() * 255 + ',' + Math.random() * 255 + ',' +
Math.random() * 255 + ')';
    document.body.style.backgroundColor = rndCol;
}

```

Mecanismos **mais recentes**, envolvem a definição dos manipuladores de evento **somente no código JavaScript**. Uma forma inicial de utilizar estes novos procedimentos, pode ser visto no exemplo abaixo:

HTML

```

<input type="button" id="btnteste" name="btntestel" value="Teste"/>

```

JavaScript

```

var btn = document.getElementById('btnteste'); //DOM
btn.onclick = bgChange;

```

O **mais novo tipo de mecanismo** de evento é definido na Especificação de Eventos Nível 2 do *Document Object Model* (DOM), que fornece aos navegadores uma nova função — **addEventListener()**. Dessa forma, o procedimento de manipulação de eventos **fica somente no código javascript e não no HTML**.

HTML

```

<input type="button" id="btnteste" name="btntestel" value="Teste"/>

```

JavaScript

```

function bgChange() {
    var rndCol = 'rgb(' + Math.random() * 255 + ',' + Math.random() * 255 + ',' +
Math.random() * 255 + ')';
    document.body.style.backgroundColor = rndCol;
}

var btn = document.getElementById('btnteste'); //DOM
btn.addEventListener('click', bgChange);

```

Dentro da função `addEventListener()`, especificamos dois parâmetros — 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.

Esse último mecanismo tem algumas vantagens sobre os mecanismos mais antigos discutidos anteriormente. Para começar, **há uma função de contraparte, `removeEventListener()`**, que remove um listener adicionado anteriormente. Por exemplo, isso removeria o listener definido no primeiro bloco de código nesta seção:

```
btn.removeEventListener('click', bgChange);
```

5.5 Informação do Objeto alvo do Evento

Às vezes, dentro de uma função de manipulador de eventos, você pode ver um parâmetro especificado com um nome como `event`, `evt`, ou simplesmente `e`. Isso é chamado de **event object**, e é passado automaticamente para os manipuladores de eventos para fornecer recursos e informações extras.

```
function bgChange(e) {
    var rndCol = 'rgb(' + Math.random() * 255 + ',' + Math.random() * 255 + ',' +
Math.random() * 255 + ')';
    document.body.style.backgroundColor = rndCol;
    alert('Objeto gerador do evento ' + e.target.getAttribute('name'))
}

var btn = document.getElementById('btnteste');
btn.addEventListener('click', bgChange);
```

5.6 Evitando o comportamento padrão

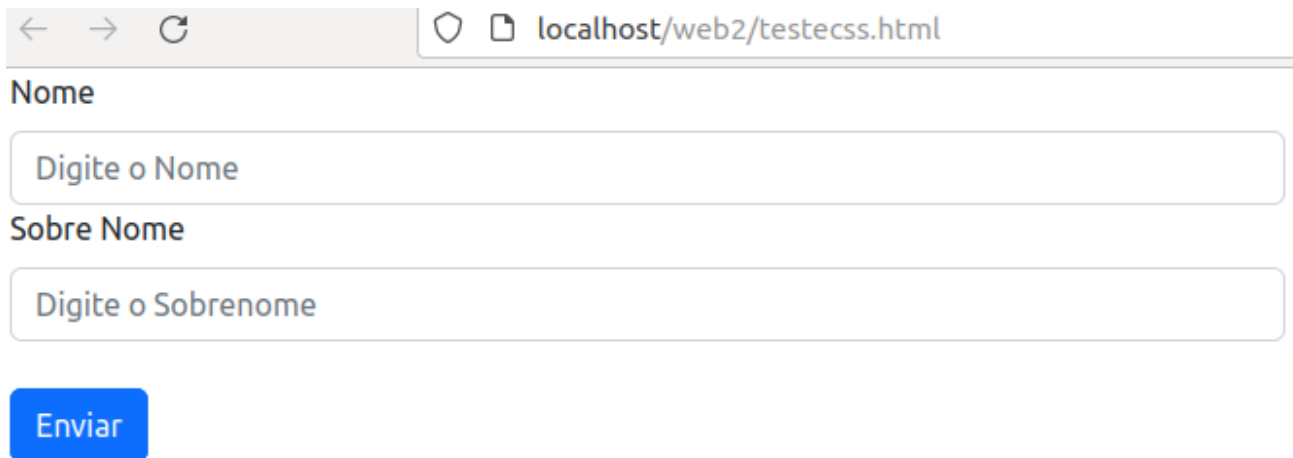
Às vezes, é necessário interromper um evento fazendo o que ele faz por padrão. O exemplo mais comum é o de um formulário da Web. Quando o usuário preenche os dados e pressiona o botão Enviar, o comportamento natural é que os dados sejam enviados para uma página específica no servidor para processamento, e o navegador seja redirecionado para uma página de "mensagem de sucesso" de algum tipo (ou a mesma página, se outra não for especificada.)

O problema surge quando o **usuário não submete os dados corretamente** - como desenvolvedor, você deve **interromper o envio** para o servidor e fornecer uma mensagem de erro informando o que está errado e o que precisa ser feito para corrigir as coisas. Alguns navegadores suportam recursos automáticos de validação de dados de formulário, mas como muitos não oferecem isso, é recomendável não depender deles e **implementar suas próprias verificações de validação**. Vamos dar uma olhada em um exemplo simples.

Na sequência é apresentada uma verificação simples dentro de um manipulador de evento **submit** (o evento submit é disparado em um formulário quando é enviado) que testa se os campos de texto estão vazios. Se estiverem, chamamos a função **preventDefault()** no objeto de evento — que

interrompe o envio do formulário — e, em seguida, exibe uma mensagem de erro no parágrafo abaixo do nosso formulário para informar ao usuário o que está errado:

VALIDAÇÃO DE DADOS EM FORMULÁRIOS



Nome

Digite o Nome

Sobre Nome

Digite o Sobrenome

Enviar

HTML

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Validação de Formulário</title>
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS v5.2.1 -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
crossorigin="anonymous">
</head>
<body>
  <div class="container-fluid">
    <form action="teste.php" method="get" id="form1">
      <label for="" class="form-label">Nome</label>
      <input type="text" class="form-control" name="nome" id="nm1">
      <label for="" class="form-label">Sobre Nome</label>
      <input type="text" class="form-control" name="sobrenome" id="snome">
      <br>
      <button type="submit" class="btn btn-primary">Enviar</button>
      <button type="button" class="btn btn-primary" id='verificar'>Verificar</button>
    </form>
  </div>
  <p id='msg'></p>
</body>
<!-- Bootstrap JavaScript Libraries -->
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"
integrity="sha384-oBqDVmMz9ATKxIep9tiCxS/Z9fNfEXiDAYTujMAeBAsjFuCZSmKbSSUnQlmh/jp3"
crossorigin="anonymous">
  </script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.min.js">
```

```

    integrity="sha384-7VPbUDkoPSGFnVtYi0QogXtr74QeVeeIs99Qfg5YCF+TidwNdjvaKZXL9NZ/e6oz"
crossorigin="anonymous">
</script>
<script src="js/validaform.js"></script>

</html>

```

CSS

```

.erro{
    border-color: red;
    box-shadow: 2px 0px 5px red;
}

```

JAVASCRIPT

```

var form1 = document.getElementById('form1');
form1.addEventListener('submit', validar);

btnv= document.getElementById('verificar');
btnv.addEventListener('click',validar);


function ver_nome(){
    var nm = document.getElementById('nm1');
    if (nm.value == ''){
        nm.classList.add('erro');
        return false;
    } else {
        nm.classList.remove('erro');
        return true;
    }
}

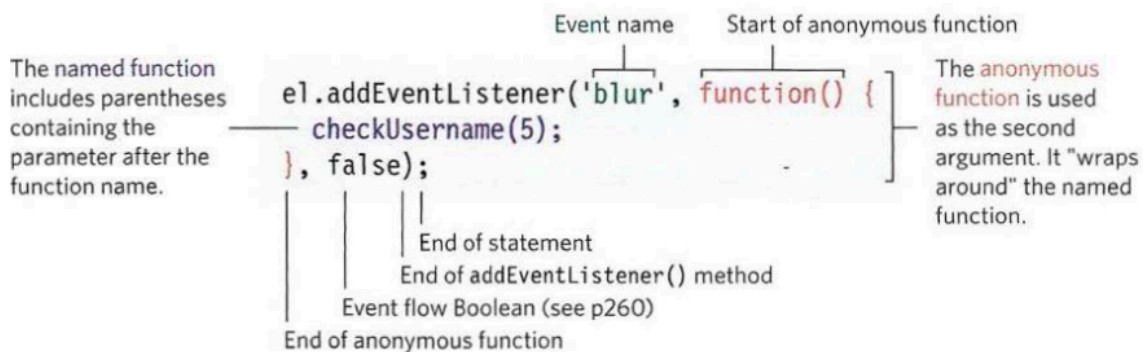
function ver_snome(){
    var snm = document.getElementById('snome');
    if (snm.value == ''){
        nm.classList.add('erro');
        return false;
    } else {
        nm.classList.remove('erro');
        return true;
    }
}

function validar(e) {
    if ((ver_nome() == false) || (ver_snome() ==false)) {
        m = document.getElementById('msg');
        m.innerHTML = 'Informações Incompletas';
        e.preventDefault();
    }
}

```

5.7 Passando parâmetros para eventos

Para realizar a passagem de parâmetros em eventos no javascript faz-se necessário utilizar as chamada **funções anônimas**. No exemplo abaixo, é utilizada esta técnica para chamar uma função `checkUsername` com um parâmetro.



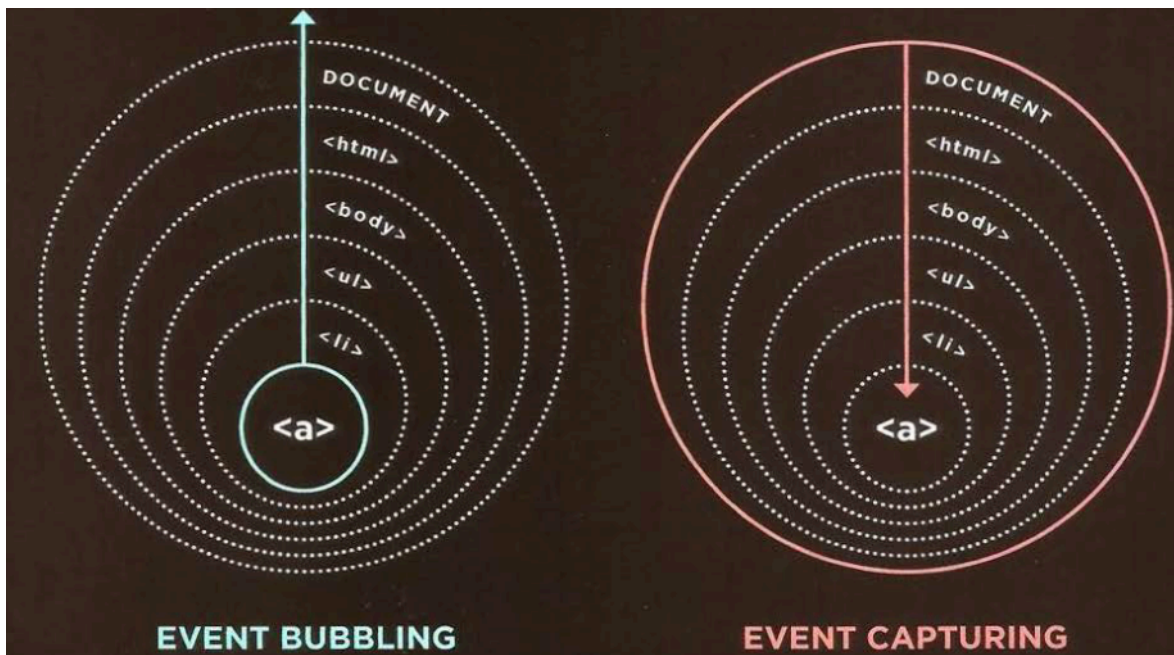
No exemplo abaixo é mostrado a utilização da passagem de parâmetros:

```
var nm = document.getElementById('nm1');
nm.addEventListener('blur', function (e) {
    checkUsername(5);
});

function checkUsername(tam) {
    var nm1 = document.getElementById('nm1');
    if (nm1.value.length < tam) {
        p = document.getElementById('msg');
        p.innerHTML = 'Tamanho do nome inválido';
    } else {
        p.innerHTML = '';
    }
}
```

5.8 Fluxo de eventos

Diversos elementos HTML podem ser aninhados formando uma estrutura hierárquia (árvore DOM). Desta forma, conforme exemplo abaixo, o click em um elemento `<a>` gera este evento em todos os elementos em que o elemento `<a>` esteja contido (seja filho). O método padrão para o fluxo destes eventos atualmente é o **“Bubbling”**.



Para evitar o comportamento padrão, pode-se usar os métodos `e.preventDefault()` e `e.stopPropagation()`.

5.9 Delegação de eventos

O fluxo (Bubbling e Capturing) de eventos no javascript nos permitem implementar um dos mais poderosos padrões de manipulação de eventos, chamado de **delegação de eventos**. A ideia é que, se tivermos muitos elementos manipulados de maneira semelhante, em vez de atribuir um manipulador a cada um deles, **colocamos um único manipulador em seu ancestral comum**. No manipulador, recebemos `event.target`, que permite identificar onde o evento realmente aconteceu e lidar com ele.

No exemplo abaixo, é implementado funcionalidade para excluir elementos de uma lista ao clicar sobre o item da lista. Observe que o evento foi implementado na lista `` e não individualmente para cada elemento ``.

HTML

```
<ul id='lista1'>
  <li>Arroz</li>
  <li>Feijão</li>
  <li>Sal</li>
</ul>
```

JavaScript

```
var lista = document.getElementById('lista1');
lista.addEventListener('click', removeitem);

function removeitem(e) {
  var target, elParent;
  target = e.target;
  elParent = target.parentNode;
```



```
    elParent.removeChild(target);  
    //target.style.textDecoration = "line-through";  
}
```

Implementação da verificação usando Orientação Objetos em JavaScript

```
class Verform{  
    constructor(nome, snome){  
        this.nome=nome;  
        this.snome=snome;  
    }  
    ver_nome(){  
        if (this.nome == '') {  
            return false;  
        } else {  
            return true;  
        }  
    }  
    ver_snome(){  
        if (this.snome == '') {  
            return false;  
        } else {  
            return true;  
        }  
    }  
}  
  
function ver_nome() {  
    var nm = document.getElementById('nm1');  
    if (nm.value == '') {  
        nm.setAttribute('class','erro');  
        return false;  
    } else {  
        nm.setAttribute('class','');  
        return true;  
    }  
}  
  
function ver_snome() {  
    var snm = document.getElementById('snome');  
    if (snm.value == '') {  
        return false;  
    } else {  
        return true;  
    }  
}  
  
function verificar(e) {  
    var nm = document.getElementById('nm1');  
    var snm = document.getElementById('snome');  
  
    let veri = new Verform(nm.value, snm.value);  
    if (veri.ver_nome() == false) {  
        p = document.getElementById('msg');
```

```
        p.innerHTML = 'Informações Incompletas';
        nm.setAttribute('class','erro');
        e.preventDefault();
    } else {
        nm.setAttribute('class','');
    }
    if (veri.ver_snome() == false){
        p = document.getElementById('msg');
        p.innerHTML = 'Informações Incompletas';
        snm.setAttribute('class','erro');
        e.preventDefault();

    } else {
        snm.setAttribute('class','');
    }
}

var form1 = document.getElementById('form1');
form1.addEventListener('submit', verificar);
```