

Javascript em ação

Eventos que transformam interfaces



Sumário

- **Introdução**

- O que são eventos?
- Por que eventos são importantes em JavaScript?

- **Eventos de Mouse**

- Click
- Double Click
- Mouse Down
- Mouse Up
- Mouse Move
- Mouse Over
- Mouse Out
- Mouse Enter
- Mouse Leave
- Context Menu

Sumário

- **Eventos de Teclado**

- Key Down
- Key Press
- Key Up

- **Eventos de Formulário**

- Submit
- Change
- Focus
- Blur
- Input
- Invalid

- **Eventos de Toque**

- Touch Start
- Touch Move

Sumário

- Touch End
 - Touch Cancel
-
- Eventos de Interface (UI)
-
- Load
 - Unload
 - Resize
 - Scroll
 - Select
-
- Eventos de Drag & Drop
-
- Drag
 - Drag Start
 - Drag End
 - Drag Over
 - Drag Enter
 - Drag Leave
 - Drop

Sumário

- **Eventos de Mídia**

- Play
- Pause
- Ended
- Time Update
- Volume Change
- Duration Change
- Loaded Metadata
- Loaded Data
- Can Play
- Can Play Through

- **Eventos de Progresso**

- Load Start
- Progress
- Load
- Abort
- Error

Sumário

- **Eventos de Animação**

- Animation Start
- Animation End
- Animation Iteration

- **Eventos de Transição**

- Transition End

- **Eventos de WebSockets**

- Open
- Message
- Error
- Close

Sumário

- **Eventos de Service Workers**

- Install
- Activate
- Fetch
- Message

- **Outros Eventos Importantes**

- Storage
- Error (em janelas e imagens)
- Offline
- Popstate
- Pagehide
- Pageshow
- Beforeunload

- **Conclusão**

- Resumo
- Como continuar aprendendo

01

Introdução

Capítulo 1: Introdução

O que são eventos?

Eventos em JavaScript representam uma forma poderosa de interagir com o ambiente do navegador e com o usuário que está interagindo com a sua página web. Eles são essencialmente sinais que são emitidos para indicar que algo significativo aconteceu. Por exemplo, quando um usuário clica em um botão, move o mouse, digita em um teclado, ou até mesmo quando uma página termina de carregar, um evento correspondente é disparado.

Esses eventos podem ser ouvidos e manipulados usando JavaScript, o que permite que programadores respondam a essas ações de maneira dinâmica. Existem vários tipos de eventos, incluindo aqueles relacionados a interações do mouse, alterações em formulários, carregamentos de página, mudanças de estado da rede, e muito mais. Cada um desses eventos pode ser capturado por meio de listeners (ouvintes) que executam funções específicas quando os eventos ocorrem, permitindo assim uma interatividade rica e personalizada nas aplicações web.

Capítulo 1: Introdução

Por que eventos são importantes em JavaScript?

Os eventos são uma parte crucial da programação em JavaScript porque permitem que as páginas web reajam a ações do usuário, tornando os sites não apenas interativos, mas também intuitivos e responsivos. Sem a capacidade de responder a eventos, as páginas web seriam estáticas e não interativas, limitando severamente a experiência do usuário.

Além de melhorar a interatividade, os eventos também oferecem uma maneira de realizar tarefas de forma eficiente. Por exemplo, um evento de 'scroll' pode ajudar a carregar conteúdo de forma dinâmica apenas quando necessário, melhorando o desempenho da página. Eventos de 'resize' da janela podem ser usados para ajustar o layout do conteúdo para diferentes tamanhos de tela, garantindo que o site seja responsivo e acessível em dispositivos variados.

Capítulo 1: Introdução

Em um nível mais avançado, os eventos são fundamentais para desenvolver aplicações web complexas que dependem de comunicações em tempo real e de manipulações sofisticadas de interface com o usuário. Por exemplo, os eventos de WebSockets permitem comunicações bidirecionais entre o cliente e o servidor sem a necessidade de recarregar a página, possibilitando a criação de experiências ricas como chats ao vivo e jogos multiplayer em tempo real.

Em resumo, entender e utilizar eventos em JavaScript é essencial para qualquer desenvolvedor web que deseja criar sites modernos, dinâmicos e reativos. Com o domínio dos eventos, você pode construir aplicações que não apenas respondem às ações dos usuários, mas também antecipam suas necessidades e melhoram sua interação geral com o site.

02

Mouse

Os eventos de mouse são uma parte fundamental da interação do usuário em qualquer aplicativo web. Eles permitem que os desenvolvedores capturem e respondam às várias ações que um usuário pode executar com um mouse, como cliques, movimentos e mais. Abaixo, exploramos os principais eventos de mouse disponíveis em JavaScript, cada um com uma breve descrição e um exemplo prático de como podem ser usados.

Eventos do Mouse

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Exemplo de Mouse Up</title>
</head>
<body>
    <button id="clickButton">Clique e solte-me</button>
    <script>
        // Acessa o elemento do botão pelo seu ID
        var element = document.getElementById('clickButton');

        // Adiciona o ouvinte de eventos para 'mouseup'
        element.addEventListener('mouseup', function() {
            // Loga a mensagem no console do navegador
            // quando o botão do mouse é liberado
            console.log('Botão do mouse liberado');
        });
    </script>
</body>
</html>
```

Eventos do Mouse

Como funciona:

Estrutura HTML: Um botão é definido com o ID clickButton.

Script JavaScript:

*O botão é acessado pelo JavaScript usando
document.getElementById('clickButton').*

Um ouvinte de eventos é adicionado ao botão para o evento mouseup. Quando o usuário clica e depois solta o botão do mouse sobre este botão, a função anexada ao ouvinte de eventos é executada.

Dentro da função, uma mensagem "Botão do mouse liberado" é escrita no console do navegador. Isso serve como uma confirmação visual de que o evento foi capturado e a função associada foi chamada corretamente.

Você pode testar este código colando-o em um arquivo HTML e abrindo-o em seu navegador. Ao clicar e soltar o botão, você verá a mensagem no console do navegador (acessível geralmente através do F12 ou clicando com o botão direito e escolhendo "Inspecionar", depois "Console").

Eventos do Mouse

1. click

O evento **click** é disparado quando o usuário clica em um elemento, ou seja, pressiona e solta o botão do mouse em um único elemento rapidamente.



```
element.addEventListener('click', function() {  
    alert('Elemento clicado!');  
});
```

2. dblclick

O evento **dblclick** ocorre quando o usuário clica duas vezes seguidas em um elemento. É útil para permitir ações rápidas de acesso ou edição.



```
element.addEventListener('click', function() {  
    alert('Elemento clicado!');  
});
```

Eventos do Mouse

3. mousedown

O evento **mousedown** é acionado no momento em que o botão do mouse é pressionado sobre um elemento. Este evento é útil para detectar o início de uma interação baseada em drag-and-drop, por exemplo.

```
element.addEventListener('mousedown', function(event) {  
  console.log('Botão do mouse pressionado:', event.button);  
  // 0 para botão esquerdo, 1 para meio, 2 para direito  
});
```

4. mouseup

O evento **mouseup** ocorre quando um botão do mouse é liberado sobre um elemento. Frequentemente usado em conjunto com **mousedown** para criar ações que exigem uma "confirmação" de clique.

```
element.addEventListener('mouseup', function() {  
  console.log('Botão do mouse liberado');  
});
```

Eventos do Mouse

5. mousemove

O evento **mousemove** é disparado sempre que o mouse é movido sobre o elemento. É extremamente útil para detectar o movimento do cursor e pode ser usado em aplicativos de desenho ou jogos.

```
element.addEventListener('mousemove', function(event) {  
  console.log('Posição do mouse:', event.clientX, event.clientY);  
});
```

6. mouseup

O evento **mouseover** acontece quando o ponteiro do mouse entra no elemento ou em um de seus filhos. É útil para destacar elementos quando o mouse passa sobre eles.

```
element.addEventListener('mouseup', function() {  
  console.log('Botão do mouse liberado');  
});
```

Eventos do Mouse

6. mouseover

O evento **mousemove** é disparado sempre que o mouse é movido sobre o elemento. É extremamente útil para detectar o movimento do cursor e pode ser usado em aplicativos de desenho ou jogos.

```
element.addEventListener('mouseover', function() {  
    console.log('Mouse sobre o elemento');  
});
```

7. mouseout

O evento **mouseout** é disparado quando o ponteiro do mouse sai do elemento ou de seus filhos. Pode ser usado para reverter mudanças feitas pelo mouseover.

```
element.addEventListener('mouseout', function() {  
    console.log('Mouse saiu do elemento');  
});
```

Eventos do Mouse

8. mouseenter

O Semelhante ao mouseover, o evento mouseenter é disparado quando o ponteiro do mouse entra no elemento, mas, diferentemente do mouseover, não se propaga (não "borbulha") para os elementos pais.

```
...  
element.addEventListener('mouseenter', function() {  
  console.log('Mouse entrou no elemento sem propagação');  
});
```

9.mouseleave

O evento mouseleave ocorre quando o mouse deixa o elemento e, assim como mouseenter, não se propaga para os elementos pais.

```
...  
element.addEventListener('mouseleave', function() {  
  console.log('Mouse deixou o elemento sem propagação');  
});
```

Eventos do Mouse

10. contextmenu

O Semelhante ao mouseover, o evento mouseenter é disparado quando o ponteiro do mouse entra no elemento, mas, diferentemente do mouseover, não se propaga (não "borbulha") para os elementos pais.

```
element.addEventListener('contextmenu', function(event) {  
    event.preventDefault();  
    // Previne o menu de contexto padrão do navegador  
    console.log('Menu de contexto acionado');  
});
```

03

Teclado

Os eventos de teclado são essenciais para interações que envolvem a entrada de dados por meio do teclado. Eles permitem que os desenvolvedores respondam a ações específicas do usuário, como pressionar e soltar teclas, e são especialmente úteis em formulários, jogos e outras funcionalidades interativas. Vamos explorar três dos principais eventos de teclado disponíveis em JavaScript.

Eventos do Teclado

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Evento Keydown</title>
</head>
<body>
  <input type="text"
    id="keyboardInput"
    placeholder="Digite algo e pressione uma tecla...">
<script>
  var inputElement = document.getElementById('keyboardInput');
  inputElement.addEventListener('keydown', function(event) {
    console.log('Tecla pressionada:', event.key);
  });
</script>
</body>
</html>
```

Eventos do Teclado

1. keydown

O evento `keydown` é disparado quando uma tecla é pressionada no teclado. Este evento ocorre antes do evento `keypress`, e é disparado independente do tipo de tecla, incluindo teclas de função que não geram caracteres como Shift, Ctrl, e Alt. O `keydown` é útil para capturar ações de teclado em um nível baixo.

```
document.addEventListener('keydown', function(event) {
  console.log('Tecla pressionada:', event.key);
  console.log('Código da tecla:', event.code);
});
```

O evento `keydown` é disparado quando uma tecla é pressionada no teclado. Este evento ocorre antes do evento `keypress`, e é disparado independente do tipo de tecla, incluindo teclas de função que não geram caracteres como Shift, Ctrl, e Alt. O `keydown` é útil para capturar ações de teclado em um nível baixo.

Eventos do Teclado

2. keypress

O evento `keypress` é disparado quando uma tecla que produz um caractere é pressionada e solta. Diferentemente de `keydown` e `keyup`, o `keypress` é ideal para capturar entradas que resultam em caracteres visíveis, ignorando outras teclas como as de controle ou navegação. Note que `keypress` está depreciado e muitos navegadores modernos recomendam usar `keydown` em seu lugar.

```
document.addEventListener('keypress', function(event) {  
  console.log('Caractere pressionado:', event.key);  
});
```

Neste código, `event.key` fornecerá o caractere associado à tecla pressionada. Apenas teclas que geram caracteres resultarão nesse evento sendo disparado.

Eventos do Teclado

3. keyup

O evento `keyup` é disparado quando uma tecla é solta no teclado. Este evento é útil para detectar o fim de uma ação de teclado e pode ser usado em combinação com `keydown` para controlar sequências de teclas ou para desativar efeitos ou funcionalidades ativadas por `keydown`.

```
document.addEventListener('keyup', function(event) {  
  console.log('Tecla liberada:', event.key);  
});
```

Neste exemplo, se você está implementando um jogo e deseja que um personagem pare de se mover quando a tecla é liberada, você pode usar o `keyup` para saber quando interromper a movimentação.

Esses eventos de teclado proporcionam controle fino sobre como os usuários interagem com sua aplicação via teclado, permitindo uma gama variada de interações diretas e responsivas.

04

Formulário

Os eventos de formulário são cruciais para interações em qualquer aplicação web, permitindo a captura e o manejo de dados inseridos pelos usuários de maneira eficiente e segura.

Eventos do Formulário

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Evento Submit de Formulário</title>
</head>
<body>
    <form id="myForm">
        <input type="text" placeholder="Nome">
        <button type="submit">Enviar</button>
    </form>
    <script>
        var formElement = document.getElementById('myForm');
        formElement.addEventListener('submit', function(event) {
            event.preventDefault(); // Impede o comportamento padrão
            console.log('Formulário enviado');
        });
    </script>
</body>
</html>
```

Eventos do Formulário

1. submit

O evento *submit* é disparado quando um formulário é enviado. Este evento é fundamental para validar dados antes que sejam enviados a um servidor.

...

```
document.getElementById('myForm').addEventListener('submit', function(event) {  
  event.preventDefault(); // Impede o envio do formulário  
  console.log('Formulário enviado');  
});
```

2. change

O evento *change* ocorre quando o valor de um elemento de entrada de dados (*<input>*, *<select>*, ou *<textarea>*) é alterado e o elemento perde o foco.

...

```
document.getElementById('mySelect').addEventListener('change', function() {  
  console.log('Opção selecionada:', this.value);  
});
```

Eventos do Formulário

3. focus

O evento *focus* é acionado quando um elemento recebe foco, seja através do clique do mouse ou navegação pelo teclado.

```
document.getElementById('myInput').addEventListener('focus', function() {  
  console.log('Input focado');  
});
```

4. blur

O evento *focus* é acionado quando um elemento perde o foco, seja através do clique do mouse ou navegação pelo teclado.

```
document.getElementById('myInput').addEventListener('blur', function() {  
  console.log('Input perdeu o foco');  
});
```

Eventos do Formulário

5. input

O evento `input` é disparado sempre que o valor de um `<input>` ou `<textarea>` é alterado, independentemente de como a alteração é feita.

```
document.getElementById('myTextarea').addEventListener('input', function() {
  console.log('Novo valor:', this.value);
});
```

6. invalid

O evento `invalid` é disparado quando um elemento de um formulário não satisfaz os critérios de validação.

```
document.getElementById('myInput').addEventListener('invalid', function() {
  console.log('Valor inválido');
});
```

05

Toque

Em dispositivos com tela sensível ao toque, os eventos de toque são fundamentais para uma interação fluida e responsiva.

Eventos do Toque

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Evento Touchstart</title>
</head>
<body>
    <div id="touchArea"
        style="width: 300px; height: 300px; background: #ccc;">
        Toque aqui
    </div>
    <script>
        var touchArea = document.getElementById('touchArea');
        touchArea.addEventListener('touchstart', function() {
            console.log('Tela tocada');
        });
    </script>
</body>
</html>
```

Eventos do Toque

1. touchstart

Disparado quando o usuário toca na tela.



```
document.addEventListener('touchstart', function(event) {  
  console.log('Tela tocada');  
});
```

2. touchmove

Acionado quando o dedo é movido sobre a tela.



```
document.addEventListener('touchmove', function(event) {  
  console.log('Movimento detectado na tela');  
});
```

Eventos do Toque

3. touchend

Ocorre quando o usuário remove o dedo da tela.

```
document.addEventListener('touchend', function(event) {  
  console.log('Toque finalizado');  
});
```

4. touchcancel

Disparado quando um toque é interrompido inesperadamente.

```
document.addEventListener('touchcancel', function(event) {  
  console.log('Toque cancelado');  
});
```

06

Interface (U)

Esses eventos são vitais para entender como os usuários interagem com o ambiente da aplicação e como a aplicação responde a mudanças no ambiente de visualização.

Eventos de Interface (UI)

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Evento Resize</title>
</head>
<body>
    <script>
        window.addEventListener('resize', function() {
            console.log('Tamanho da janela alterado para', window.innerWidth, 'x',
window.innerHeight);
        });
    </script>
</body>
</html>
```

Eventos de Interface (UI)

1. load

O evento *load* é disparado quando um objeto, como uma página da web ou uma imagem, foi completamente carregado.



```
window.addEventListener('load', function() {  
    console.log('Página carregada');  
});
```



2. unload

Ocorre quando uma página está sendo descarregada.



```
window.addEventListener('unload', function() {  
    console.log('Página sendo descarregada');  
});
```

Eventos de Interface (UI)

3. resize

Disparado quando a janela do navegador é redimensionada.

```
...  
  
window.addEventListener('resize', function() {  
  console.log('Janela redimensionada para', window.innerWidth, 'por',  
  window.innerHeight);  
});
```

4. scroll

O evento scroll é acionado quando ocorre um deslocamento na área visível de um elemento.

```
...  
  
window.addEventListener('scroll', function() {  
  console.log('Página rolada');  
});
```

Eventos de Interface (UI)

5. select

Disparado quando o texto é selecionado em um elemento como <input> ou <textarea>.

```
document.getElementById('myTextarea').addEventListener('select', function() {  
    console.log('Texto selecionado');  
});
```

07

Drag & Drop

Os eventos de Drag & Drop permitem ao usuário arrastar um objeto de um ponto a outro na interface. Esses eventos são essenciais para interfaces onde a manipulação direta de objetos é necessária, como em aplicações de design, jogos ou ferramentas de organização.

Eventos de Drag & Drop

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Evento Drag & Drop</title>
</head>
<body>
    <div id="dropZone"
        style="width: 300px; height: 300px; background: lightblue;">
        Arraste e solte algo aqui
    </div>
    <script>
        var dropZone = document.getElementById('dropZone');
        dropZone.addEventListener('dragover', function(event) {
            event.preventDefault(); // Necessário para permitir o evento drop
        });
        dropZone.addEventListener('drop', function(event) {
            event.preventDefault();
            console.log('Objeto solto');
        });
    </script>
</body>
</html>
```

Eventos de Drag & Drop

1. drag

Disparado quando um elemento é arrastado.

```
element.addEventListener('drag', function(event) {  
    console.log('Elemento sendo arrastado');  
});
```

2. dragstart

Acionado no início da operação de arrasto, útil para preparar dados ou ajustar a interface.

```
element.addEventListener('dragstart', function(event) {  
    console.log('Arrasto iniciado');  
});
```

Eventos de Drag & Drop

3. dragend

Ocorre quando a operação de arrasto é finalizada, independentemente de ser concluída ou cancelada.

```
element.addEventListener('dragend', function(event) {  
  console.log('Arrasto finalizado');  
});
```

4. dragover

Disparado quando um elemento arrastável é movido sobre um alvo de soltura, permitindo manipular a aparência do alvo.

```
element.addEventListener('dragover', function(event) {  
  event.preventDefault();  
  // Necessário para permitir a soltura  
  console.log('Elemento sobre alvo de soltura');  
});
```

Eventos de Drag & Drop

5. dragenter

Acionado quando um elemento arrastável entra em um alvo de soltura, útil para fornecer feedback visual.

```
element.addEventListener('dragenter', function(event) {  
  console.log('Elemento entrou no alvo de soltura');  
});
```

6. dragleave

Ocorre quando um elemento arrastável sai de um alvo de soltura, revertendo mudanças visuais aplicadas no dragenter.

```
element.addEventListener('dragleave', function(event) {  
  console.log('Elemento saiu do alvo de soltura');  
});
```

Eventos de Drag & Drop

7. drop

Disparado quando um elemento é solto em um alvo de soltura, é o momento de finalizar a transferência de dados ou ajustar a interface com base no resultado do arrasto.

```
element.addEventListener('drop', function(event) {  
    event.preventDefault(); // Impede a ação padrão  
    console.log('Elemento solto');  
});
```

08

Mídia

Eventos de mídia são utilizados para controlar e responder a interações com conteúdo de áudio e vídeo, permitindo a criação de experiências ricas de multimídia interativa.

Eventos de Mídia

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Eventos de Mídia</title>
</head>
<body>
    <video id="myVideo" width="320" height="240" controls>
        <source src="movie.mp4" type="video/mp4">
            Seu navegador não suporta vídeo HTML5.
    </video>
    <script>
        var video = document.getElementById('myVideo');
        video.addEventListener('play', function() {
            console.log('Vídeo iniciado');
        });
        video.addEventListener('pause', function() {
            console.log('Vídeo pausado');
        });
    </script>
</body>
</html>
```

Eventos de Mídia

1. play

Acionado quando a reprodução de mídia é iniciada, seja por uma ação do usuário ou programaticamente.

```
mediaElement.addEventListener('play', function() {  
    console.log('Reprodução iniciada');  
});
```

2. pause

Ocorre quando a reprodução de mídia é pausada.

```
mediaElement.addEventListener('pause', function() {  
    console.log('Reprodução pausada');  
});
```

Eventos de Mídia

3. ended

Disparado quando a mídia termina de ser reproduzida até o fim.

```
● ● ●  
  
mediaElement.addEventListener('ended', function() {  
  console.log('Reprodução finalizada');  
});
```

4. timeupdate

Acionado quando a posição de reprodução da mídia é atualizada, útil para atualizar interfaces de usuário como barras de progresso.

```
● ● ●  
  
mediaElement.addEventListener('timeupdate', function() {  
  console.log('Posição de tempo atualizada');  
});
```

Eventos de Mídia

5. volumechange

Ocorre quando o volume da mídia é alterado, seja pelo usuário ou programaticamente.

```
mediaElement.addEventListener('volumechange', function() {  
    console.log('Volume alterado');  
});
```

6. durationchange

Disparado quando a duração total da mídia é alterada, como ao carregar um novo vídeo.

```
mediaElement.addEventListener('durationchange', function() {  
    console.log('Duração da mídia alterada');  
});
```

Eventos de Mídia

7. loadedmetadata

Acionado quando os metadados da mídia, como dimensões e duração, são carregados.

```
● ● ●  
mediaElement.addEventListener('loadedmetadata', function() {  
    console.log('Metadados carregados');  
});
```

8. loadeddata

Ocorre quando os primeiros dados da mídia são carregados, indicando que a mídia pode começar a ser reproduzida.

```
● ● ●  
mediaElement.addEventListener('loadeddata', function() {  
    console.log('Dados da mídia carregados');  
});
```

Eventos de Mídia

9. canplay

Acionado quando a mídia pode ser reproduzida até o fim sem interrupção, assumindo que a taxa de download permaneça constante.

```
mediaElement.addEventListener('canplay', function() {  
    console.log('Mídia pronta para ser reproduzida sem  
    interrupções');  
});
```

10. canplaythrough

Ocorre quando a mídia pode ser reproduzida até o fim sem interrupção, assumindo que a taxa de download permaneça constante.

```
mediaElement.addEventListener('canplaythrough', function() {  
    console.log('Mídia pronta para reprodução contínua até o fim');  
});
```

09

Progresso

Eventos de progresso são utilizados para monitorar o progresso do carregamento de recursos na página, permitindo aos desenvolvedores criar interfaces de usuário que informam aos usuários sobre o estado do carregamento de conteúdos.

Eventos de Progresso

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Eventos de Progresso</title>
</head>
<body>
    <script>
        var xhr = new XMLHttpRequest();
        xhr.open('GET', 'your-file.txt', true);

        xhr.addEventListener('loadstart', function() {
            console.log('Início do carregamento');
        });

        xhr.addEventListener('progress', function(event) {
            if (event.lengthComputable) {
                var percentComplete = (event.loaded / event.total) * 100;
                console.log('Progresso do carregamento: ' +
percentComplete.toFixed(2) + '%');
            }
        });

        xhr.addEventListener('load', function() {
            if (xhr.status === 200) {
                console.log('Arquivo carregado com sucesso');
            }
        });

        xhr.send();
    </script>
</body>
</html>
```

Eventos de Progresso

1. loadstart

Disparado quando o navegador começa a carregar um recurso

```
document.addEventListener('loadstart', function(event) {  
  console.log('Início do carregamento');  
});
```

2. progress

Acionado para indicar o progresso do carregamento de um recurso.

```
document.addEventListener('progress', function(event) {  
  console.log('Progresso do carregamento:', event.loaded, '/',  
  event.total);  
});
```

Eventos de Progresso

3. load

Ocorre quando um recurso foi completamente carregado.


```
window.addEventListener('load', function() {  
  console.log('Carregamento completo');  
});
```

4. abort

Disparado quando o carregamento de um recurso é abortado.



```
document.addEventListener('abort', function() {  
  console.log('Carregamento abortado');  
});
```

Eventos de Progresso

5. error

Acionado quando ocorre um erro durante o carregamento de um recurso.

```
...  
  
window.addEventListener('error', function(event) {  
  console.log('Erro durante o carregamento', event);  
});
```

10

Animação

Eventos de animação são utilizados para gerenciar animações CSS, permitindo aos desenvolvedores capturar momentos chave do ciclo de vida de uma animação, como seu início, repetição e conclusão.

Eventos de Animação

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Eventos de Animação</title>
    <style>
        @keyframes example {
            from {background-color: red;}
            to {background-color: yellow;}
        }
        #animatedBox {
            width: 100px;
            height: 100px;
            background-color: red;
            animation-name: example;
            animation-duration: 4s;
            animation-iteration-count: infinite;
        }
    </style>
</head>
<body>
    <div id="animatedBox"></div>
    <script>
        var box = document.getElementById('animatedBox');
        box.addEventListener('animationstart', function() {
            console.log('Animação iniciada');
        });
        box.addEventListener('animationend', function() {
            console.log('Animação finalizada');
        });
        box.addEventListener('animationiteration', function() {
            console.log('Nova iteração da animação iniciada');
        });
    </script>
</body>
</html>
```

Eventos de Animação

1. animationstart

Disparado quando uma animação CSS começa.



```
element.addEventListener('animationstart', function() {  
  console.log('Animação iniciada');  
});
```



```
element.addEventListener('animationend', function() {  
  console.log('Animação finalizada');  
});
```

Eventos de Animação

3. animationiteration

Acionado a cada ciclo de uma animação, exceto o último.

...

```
element.addEventListener('animationiteration', function() {  
  console.log('Nova iteração da animação começou');  
});
```

11

Transição

Eventos de transição são cruciais para capturar o fim de uma transição CSS, permitindo reações dinâmicas a mudanças estilísticas que dependem do tempo.

Eventos de Transição

Exemplo:

```
● ● ●

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Evento de Transição</title>
    <style>
        #transitionBox {
            width: 100px;
            height: 100px;
            background-color: blue;
            transition: width 2s;
            /* Transição aplicada à largura */
        }
        #transitionBox:hover {
            width: 200px;
            /* Largura do elemento quando o mouse passa por cima */
        }
    </style>
</head>
<body>
    <div id="transitionBox"></div>
    <script>
        var box = document.getElementById('transitionBox');
        box.addEventListener('transitionend', function(event) {
            console.log('Transição finalizada para a propriedade',
event.propertyName);
        });
    </script>
</body>
</html>
```

Eventos de Transição

1. transitionend

Disparado quando uma transição CSS termina. Se várias propriedades são animadas, este evento ocorre para cada propriedade.

```
element.addEventListener('transitionend', function() {  
  console.log('Transição finalizada');  
});
```

12

WebSockets

WebSockets proporcionam um canal de comunicação bidirecional e em tempo real entre o navegador e o servidor, e os eventos de WebSocket ajudam a gerenciar essa comunicação.

Eventos de WebSockets

Exemplo:

• • •

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Eventos de WebSocket</title>
</head>
<body>
    <script>
        var socket = new WebSocket('wss://echo.websocket.org');
        // Usando um servidor de echo público para exemplo
        socket.addEventListener('open', function() {
            console.log('Conexão WebSocket aberta');
            socket.send('Hello, server!');
            // Envia uma mensagem ao servidor
        });
        socket.addEventListener('message', function(event) {
            console.log('Mensagem recebida:', event.data);
            // Recebe e exibe a mensagem do servidor
        });
        socket.addEventListener('close', function() {
            console.log('Conexão WebSocket fechada');
        });
        socket.addEventListener('error', function(error) {
            console.log('Erro na WebSocket:', error);
        });
    </script>
</body>
</html>
```

Eventos de WebSockets

1. open

Disparado quando uma conexão WebSocket é aberta.

```
websocket.addEventListener('open', function() {  
  console.log('Conexão WebSocket aberta');  
});
```

2. message

Disparado quando uma mensagem é recebida através de uma conexão WebSocket.

```
websocket.addEventListener('message', function(event) {  
  console.log('Mensagem recebida:', event.data);  
});
```

Eventos de WebSockets

3. error

Acionado quando ocorre um erro na conexão WebSocket.

```
websocket.addEventListener('error', function() {  
    console.log('Erro na WebSocket');  
});
```

4. close

Disparado quando uma conexão WebSocket é fechada, seja por uma ação do usuário, por um erro, ou pelo servidor.

```
websocket.addEventListener('close', function() {  
    console.log('Conexão WebSocket fechada');  
});
```

13

Service Workers

Os eventos de Service Worker são fundamentais para gerenciar o ciclo de vida e as funcionalidades de um Service Worker, permitindo que aplicações funcionem offline e melhorem o desempenho do carregamento de recursos.

Eventos de WebSockets

Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Eventos de Service Worker</title>
</head>
<body>
    <script>
        if ('serviceWorker' in navigator) {
            navigator.serviceWorker.register('/sw.js') // O arquivo sw.js deve
estar na raiz do servidor
                .then(function(registration) {
                    console.log('Service Worker registrado com sucesso com escopo:', registration.scope);
                })
                .catch(function(error) {
                    console.log('Falha ao registrar o Service Worker:', error);
                });
        }

        navigator.serviceWorker.addEventListener('message', function(event) {
            console.log('Mensagem recebida do Service Worker:', event.data);
        });
    </script>
</body>
</html>
```

Eventos de WebSockets

1. install

Disparado quando o Service Worker está sendo instalado. É um momento oportuno para cache de recursos.

```
• • •
```

```
self.addEventListener('install', function(event) {  
  console.log('Service Worker instalado');  
});
```

2. activate

Acionado quando o Service Worker é ativado, o que ocorre após a instalação ou atualização. Utilizado para limpeza de caches antigos.

```
• • •
```

```
self.addEventListener('activate', function(event) {  
  console.log('Service Worker ativado');  
});
```

Eventos de WebSockets

3. fetch

Disparado para cada solicitação de rede feita pela página, permitindo que o Service Worker responda com recursos personalizados, possivelmente cacheados.

...

```
self.addEventListener('fetch', function(event) {  
  console.log('Solicitação de rede:', event.request.url);  
});
```

4. message

Ocorre quando o Service Worker recebe uma mensagem, geralmente enviada por páginas que ele controla.

...

```
self.addEventListener('message', function(event) {  
  console.log('Mensagem recebida:', event.data);  
});
```

14

Outros

Além dos eventos específicos já mencionados, há uma variedade de outros eventos que desempenham papéis vitais em contextos específicos, melhorando a interatividade, a segurança e a usabilidade das aplicações web.

Outros Eventos

1. storage

Disparado quando um armazenamento local é modificado (LocalStorage ou SessionStorage).

```
...  
  
window.addEventListener('storage', function(event) {  
  console.log('Dados alterados no storage:', event.key);  
});
```

2. error (em janelas e imagens)

Acionado quando ocorrem erros durante o carregamento de imagens ou arquivos JavaScript, útil para tratamento de erros e relatórios.

```
...  
  
window.addEventListener('error', function(event) {  
  console.log('Erro ao carregar:', event.filename);  
});
```

Outros Eventos

3. online e offline

Disparados quando o navegador detecta uma mudança no estado da conexão da internet, ajudando a adaptar a aplicação conforme a disponibilidade da rede.

```
...  
  
window.addEventListener('online', function() {  
  console.log('Conexão com a internet restabelecida');  
});  
window.addEventListener('offline', function() {  
  console.log('Conexão com a internet perdida');  
});
```

4. popstate

Ocorre quando o estado ativo da sessão de histórico é modificado, por exemplo: Quando o usuário clica nos botões de navegação de página (avançar e voltar).

```
...  
  
window.addEventListener('popstate', function(event) {  
  console.log('O estado do histórico mudou');  
});
```

Outros Eventos

5. pagehide e pageshow

Eventos que são acionados quando uma página está sendo ocultada ou mostrada, respectivamente, úteis em aplicações de página única para controlar transições ou animações.

```
...  
  
window.addEventListener('pagehide', function(event) {  
    console.log('Página ocultada');  
});  
window.addEventListener('pageshow', function(event) {  
    console.log('Página mostrada');  
});
```

6. beforeunload

Disparado antes da janela ou aba ser fechada, permitindo que a aplicação solicite uma confirmação do usuário se há dados que não foram salvos.

```
...  
  
window.addEventListener('beforeunload', function(event) {  
    event.returnValue = 'Tem certeza que deseja sair?';  
    console.log('Tentativa de fechar a janela');  
});
```

15

Conclusão

Este eBook ofereceu uma visão abrangente dos eventos em JavaScript, explicando como cada um pode ser utilizado para enriquecer a interatividade e funcionalidade das aplicações web. Aprendemos sobre a variedade de eventos disponíveis, desde interações básicas do usuário, como cliques e entradas de teclado, até processos mais complexos, como comunicações via WebSockets e manipulações via Service Workers.

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, e diagramado por humano.

O conteúdo gerado foi por fins didáticos de construção, não realizado uma validação cuidadosamente humana no conteúdo e pode conter erros gerados por uma IA

Contatos: [Instagram](#) - [Github](#) - [LinkedIn](#)