MATERIAL EXTRA

Eventos do DOM

growdev

EVENTOS DO DOM

A integração do Javascript com o DOM permite que uma página web estática se torne em uma aplicação dinâmica, uma vez que o conteúdo do documento pode ser alterado em tempo real através de ações do código executado. Contudo, a interatividade proporcionada pelo Javascript não se limita apenas a comandos do código para o documento. É possível, também, fazer com que o próprio **usuário** da aplicação se comunique com a aplicação e **execute comandos**.

As **ações do usuário** como o clique de um botão podem ser programadas para que a página mude ou execute determinada rotina. Para capturar essas ações, o Javascript fornece uma interface de **eventos**: um conjunto de ações disponíveis para que o usuário tenha interação com a página e para que o programa possa **reagir** de alguma forma.

Sempre que uma ação é feita na página, o evento correspondente é **disparado**. É um tipo de sinalização da página ao programa Javascript que pode então realizar um tratamento específico, de acordo com o sinal enviado. Alguns exemplos de ações do usuário na página podem ser:

- o clique no mouse;
- o aperto de uma tecla do teclado;
- o preenchimento de um campo de texto na página;
- o envio de um formulário;
- o redimensionamento da janela do navegador;
- a rolagem da página (scroll);
- entre outros.

Ou seja, podemos implementar diferentes códigos que são executados à medida em que determinada ação do usuário é observada. Esta

característica faz com que o Javascript ofereça uma interface de programação **event-driven** (baseada em eventos).

Principais eventos

Existem diversos eventos que podem ser capturados pelo Javascript. Alguns deles possuem utilização bastante frequente, independentemente do tipo da aplicação. Abaixo estão listados alguns dos principais eventos disponíveis.

Tipo	Usado para capturar...	Ex. de aplicação
click	O clique do mouse sobre algum componente, seja um <button>, uma ou até mesmo um <h1>.	Adição de comportamento ao clique de botões, como no caso de um botão que abre um menu.
change	Quando um elemento tem seu valor alterado. Aplica-se a elementos do tipo <input> para validar as alterações.	Validação de CPF após alteração do valor do campo.
keyup	Quando o usuário pressiona e solta uma tecla no teclado. Aplica-se a todos os componentes mas é geralmente usado em <input>.	Mascaramento de CPF conforme o campo é digitado (a cada caractere).
submit	Quando um formulário HTML é submetido.	Validação de todos os campos de um formulário.
focus	Quando um elemento é selecionado (recebe foco), seja através de um clique, ou de forma forçada (comando JS) ou então através da troca via TAB.	Estilização de um componente para indicar que ele foi selecionado.
load	O momento em que o documento HTML acabou de ser carregado	Inicialização de componentes na página e pré-renderização como telas de loading.

O principal evento é, com certeza, o `click`. Com ele é possível adicionar comportamento a qualquer elemento da tela quando é **clicado**. Os exemplos mais comuns são a abertura e o fechamento de menus após o clique no botão.

Alguns eventos são mais aplicáveis a **formulários** HTML e seus campos, como é o caso do `change` e do `keyup`. Suas aplicações são frequentemente voltadas à validação dos valores informados em componentes do tipo `<input>`, além do mascaramento de textos.

Outros eventos podem ser menos utilizados mas são extremamente úteis para a página. O evento `load` é disparado sempre que a página é carregada. Com o `load` é possível, por exemplo, preparar a página para o carregamento das informações usando uma tela de loading provisória.

Para um maior detalhamento dos eventos mencionados acima, bem como uma listagem de todos os eventos disponíveis e suas funções, o link Event reference | MDN pode ser bastante útil.

Captura

Eventos estão sempre ocorrendo dentro de uma aplicação web, mesmo que não sejam capturados. Para que seja possível adicionar um **tratamento** a algum evento, precisamos indicar o tipo do evento além de uma função a ser executada. Existem duas principais formas de capturar eventos no JS: **event listener** e **HTML inline**.

Event Listener

O método de captura via **event listener** é feito através de um script Javascript, em que selecionamos um elemento e adicionamos um

"listener". Como o nome sugere, o listener fica "escutando" (ou, então, observando) a página para identificar sempre que determinado evento ocorre.

A função usada para definir um listener em um elemento é a `addEventListener`, que recebe dois parâmetros: o evento a ser observado e a função que deve ser executada sempre que o evento ocorrer. Considere o HTML e o código Javascript abaixo.

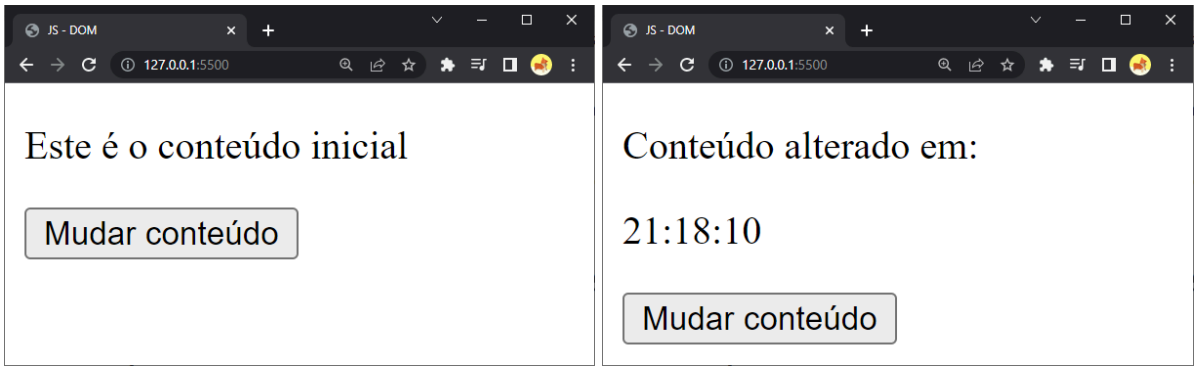
```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - Events</title>
  </head>
  <body>
    <div id="content">
      <p>Este é o conteúdo inicial</p>
    </div>

    <button id="changeContent">Mudar conteúdo</button>

    <script src="./script.js"></script>
  </body>
</html>
```

No HTML acima está definida uma tag `<div>` com ID "content", representando uma seção de conteúdo da página. Abaixo da `<div>` existe um botão que servirá para que o conteúdo seja alterado dinamicamente. Para que o conteúdo mude no clique do botão é preciso escutar o evento.

```
const buttonElement = document.querySelector("#changeContent");

function changeContentText() {
  const content = document.querySelector("#content");
  content.innerHTML = "<p>Conteúdo alterado em: ";
  content.innerHTML += new Date().toLocaleTimeString();
  content.innerHTML += "</p>";
}

buttonElement.addEventListener("click", changeContentText);
```

O código acima primeiramente seleciona o elemento DOM que representa o botão onde queremos observar o evento. Logo após, a função `changeContentText` é declarada. Seu objetivo é selecionar o elemento do conteúdo e mudar o seu valor usando a hora atual como parte do texto.

Por último, a função `addEventListener` é chamada dentro do elemento do botão. Os parâmetros passados são "click" – o evento que deve ser escutado – e a função a ser executada. Em resumo, o código Javascript definido acima irá trocar o conteúdo sempre que o botão for clicado.



Página antes e depois de um clique no botão "changeContent"

Note que passamos apenas o nome da função (sem parênteses) na chamada do `addEventListener`, pois o segundo parâmetro do listener é a referência a função que será executada "on demand". Também é possível passar uma função anônima ou uma arrow function como parâmetro.

```
// Usando função anônima
buttonElement.addEventListener("click", function() {
  const content = document.querySelector("#content");
  content.innerHTML = "<p>Conteúdo alterado em: ";
  content.innerHTML += new Date().toLocaleTimeString();
  content.innerHTML += "</p>";
});

// Usando arrow function
buttonElement.addEventListener("click", () => {
  const content = document.querySelector("#content");
  content.innerHTML = "<p>Conteúdo alterado em: ";
  content.innerHTML += new Date().toLocaleTimeString();
  content.innerHTML += "</p>";
});

// Usando arrow function para chamar a função previamente criada
buttonElement.addEventListener("click", () => changeContentText());
```

Inline

Outra forma de capturar um evento é o adicionando diretamente como **atributo** de um elemento HTML. Para estes casos, os eventos são nomeados com um **prefixo** "on". Ou seja, o evento `click` passa a ser referido como `onclick`. Considere o mesmo exemplo acima, mas com o evento sendo capturado diretamente via atributo `onclick`.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - Events</title>
  </head>
  <body>
    <div id="content">
      <p>Este é o conteúdo inicial</p>
    </div>

    <button id="changeContent" onclick="changeContentText()">
      Mudar conteúdo
    </button>

    <script src="./script.js"></script>
  </body>
</html>
```

Perceba que a definição da função permanece dentro do arquivo script.js, e apenas chamamos ela no evento `onclick` do botão. Também é possível executar comandos diretamente no elemento, sem necessidade de um arquivo Javascript externo.

No código acima, o resultado é o mesmo da função executada via event listener. Em termos práticos não há nenhuma diferença visto que o resultado para a aplicação é o mesmo. Em muitos casos, a escolha entre um método e outro é apenas por questões de organização.

Ainda sobre inline event, é possível adicionar os atributos via Javascript ao invés de defini-los diretamente no HTML. Ainda que os eventos sejam definidos via Javascript, esta abordagem não é a mesma do event listener

e a nomenclatura segue com o prefixo “on”. Os eventos estão anexados como propriedades do próprio elemento.

```
const buttonElement = document.querySelector("#changeContent");

function changeContentText() {
  const content = document.querySelector("#content");
  content.innerHTML = "<p>Conteúdo alterado em: ";
  content.innerHTML += new Date().toLocaleTimeString();
  content.innerHTML += "</p>";
}

buttonElement.onclick = changeContentText;
```

Ou seja, com inline events é possível adicionar um listener de evento diretamente em uma tag HTML ou então setar no element via Javascript.

Argumento de evento

Note no exemplo anterior que a função usada pelo event listener – chamada de `changeContentText` – não possui parâmetros. Isto é, uma função de evento não requer parâmetros para que seja executada.

Porém, invariavelmente os eventos passam como parâmetro um **argumento de evento** que as funções podem obter. Esse argumento é um objeto que fornece **informações** sobre o evento disparado. Essas informações podem ser úteis para a execução do código da função.

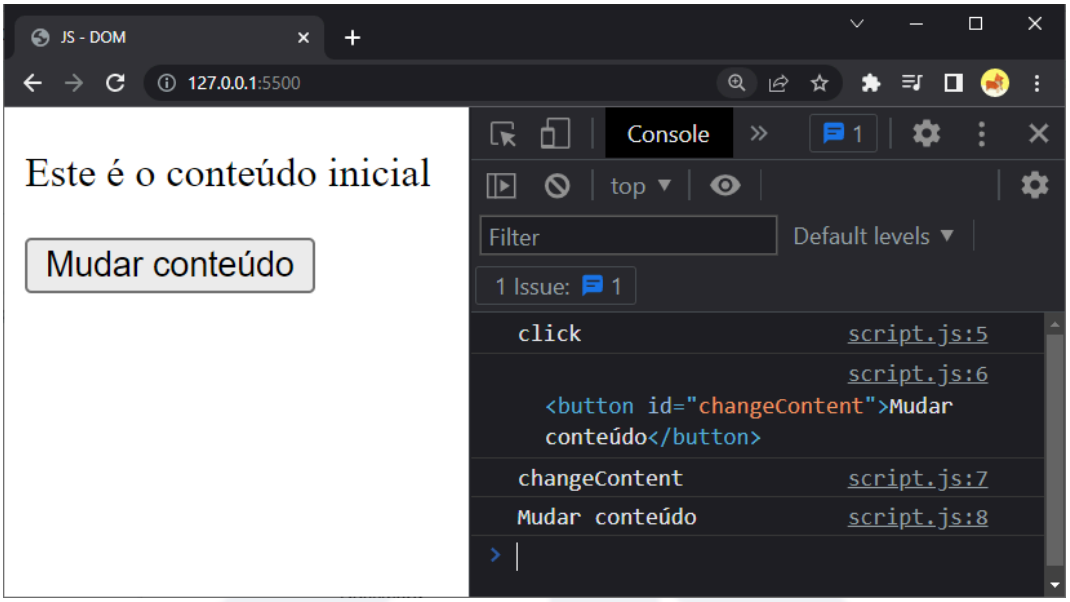
Cada evento possui propriedades específicas, mas todos compartilham de duas informações básicas: **type** e **target**. A propriedade `type` indica

qual foi o evento disparado (útil quando uma mesma função executa para mais de um tipo de evento).

Já o `target` indica qual elemento disparou o evento. O `target` armazena um item `element` do DOM. Isto é, a partir dele conseguimos obter não apenas o nome do elemento que disparou o evento, mas também o seu ID, seu `innerHTML`, seu `value` (em caso de `<input>`), bem como todas as demais propriedades de um `element`. Considere o mesmo HTML do exemplo anterior, mas com a seguinte mudança no event listener do botão:

```
const buttonElement = document.querySelector("#changeContent");

function getEventInfo(event) {
  console.log(event.type);
  console.log(event.target);
  console.log(event.target.id);
  console.log(event.target.innerHTML);
}

buttonElement.addEventListener("click", getEventInfo);
```

A função `getEventInfo` recebe, então, uma variável chamada `event`. Usamos esta função no listener do clique no botão “changeContent”. O objetivo da função é mostrar as propriedades do evento disparado bem como do elemento que sofreu o disparo – neste caso, o botão.



O nome "event" para a variável parte de uma convenção bastante utilizada, mas não é definitivo. Podemos usar qualquer outro nome, desde que esteja na primeira posição de parâmetros da função. Outra forma bastante usada é apenas "e" como nome do argumento.

Nos casos onde a abordagem escolhida é o inline event através de uma tag HTML, devemos fazer a chamada a função passando `event` como parâmetro. Neste caso, o nome "event" deve ser usado obrigatoriamente na chamada pois indica o evento dentro da tag.

```
<button id="changeContent" onclick="getEventInfo(event)">
  Mudar conteúdo
</button>
```

PreventDefault

Muitos componentes HTML possuem um **comportamento padrão** já

definido, disparando ações automaticamente quando um evento ocorre. Por exemplo, tags `<a>` navegam para a página definida no atributo `href` sempre que seu conteúdo é clicado. Outro exemplo é um formulário HTML que tem seu conteúdo submetido no evento `submit`.

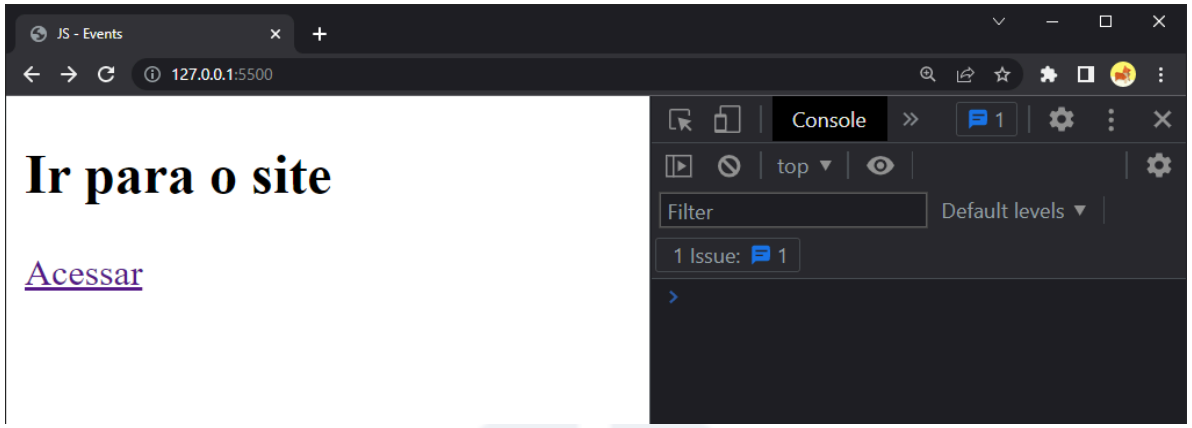
Porém, existem situações em que o comportamento padrão não é o que queremos. Podemos imaginar um exemplo de tag `<a>` onde o site especificado no `href` só pode ser acessado se determinada condição for satisfeita.

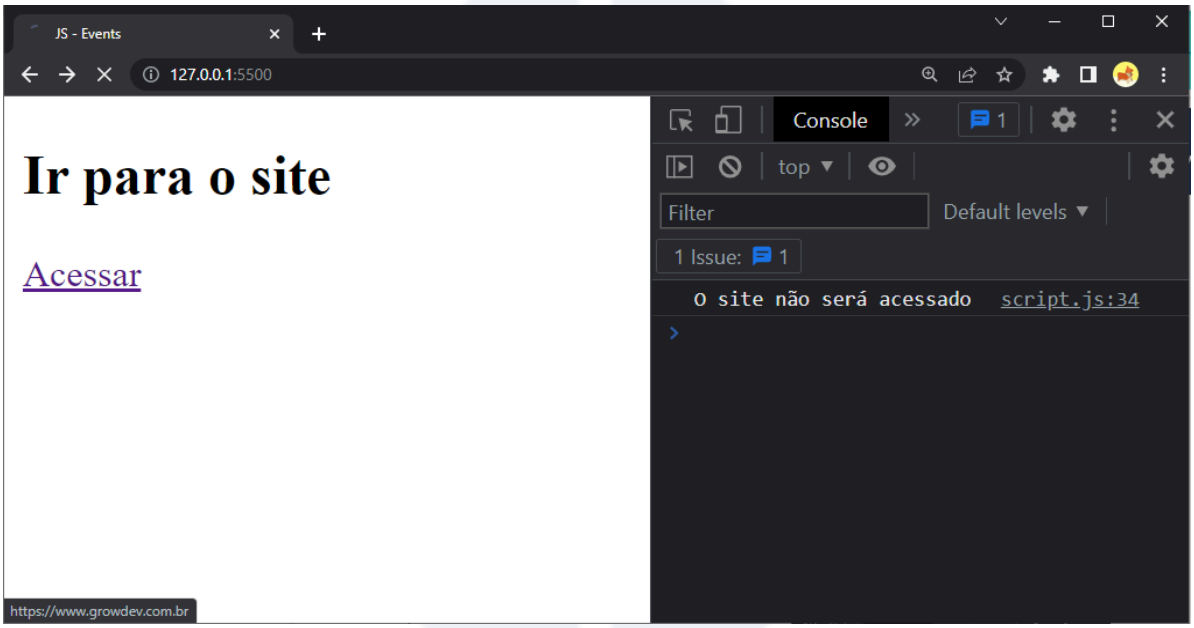
```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>JS - Events</title>
  </head>
  <body>
    <h2>Ir para o site</h2>
    <a id="linkGrowdev" href="https://www.growdev.com.br">Acessar</a>

    <script src="./script.js"></script>
  </body>
</html>
```

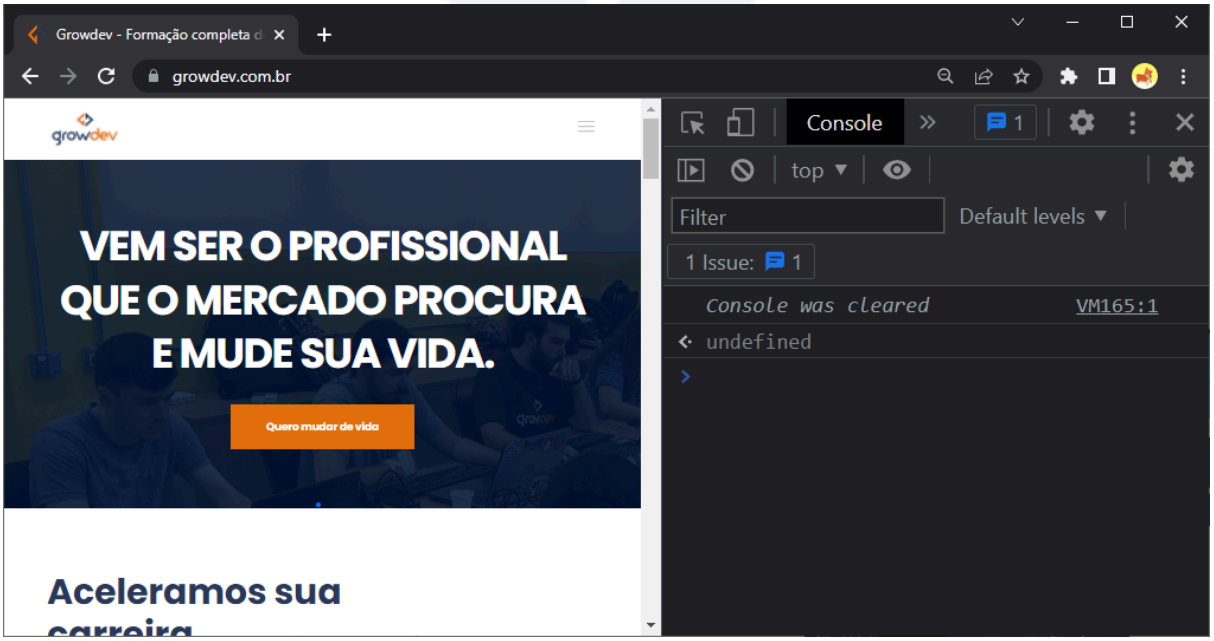
```
const link = document.querySelector("#linkGrowdev");
const someCondition = false;

link.addEventListener("click", () => {
  if (someCondition) {
    alert("Acessando o site da Growdev ... ");
    window.location.assign("https://www.growdev.com.br");
  } else {
    console.log("O site não será acessado");
  }
});
```

No exemplo acima, definimos uma tag <a> que possui o `href` para o site da Growdev. Também definimos no Javascript um event listener para o evento `click` da tag que direciona para o site da Growdev se, e somente se, a condição `someCondition` for `true`.

Perceba que `someCondition` é `false`, por isso o resultado do event listener em tese deveria ser o não direcionamento para o site da Growdev. Ainda assim, o **comportamento padrão** de uma tag <a> é direcionar para o que está no atributo `href`. Por isso, o código acima resultará no **acesso ao site**.


JS - Events
127.0.0.1:5500
Ir para o site
Acessar
Console
top
Filter
Default levels
1 Issue: 1


JS - Events
127.0.0.1:5500
Ir para o site
Acessar
https://www.growdev.com.br
Console
top
Filter
Default levels
1 Issue: 1
O site não será acessado script.js:34

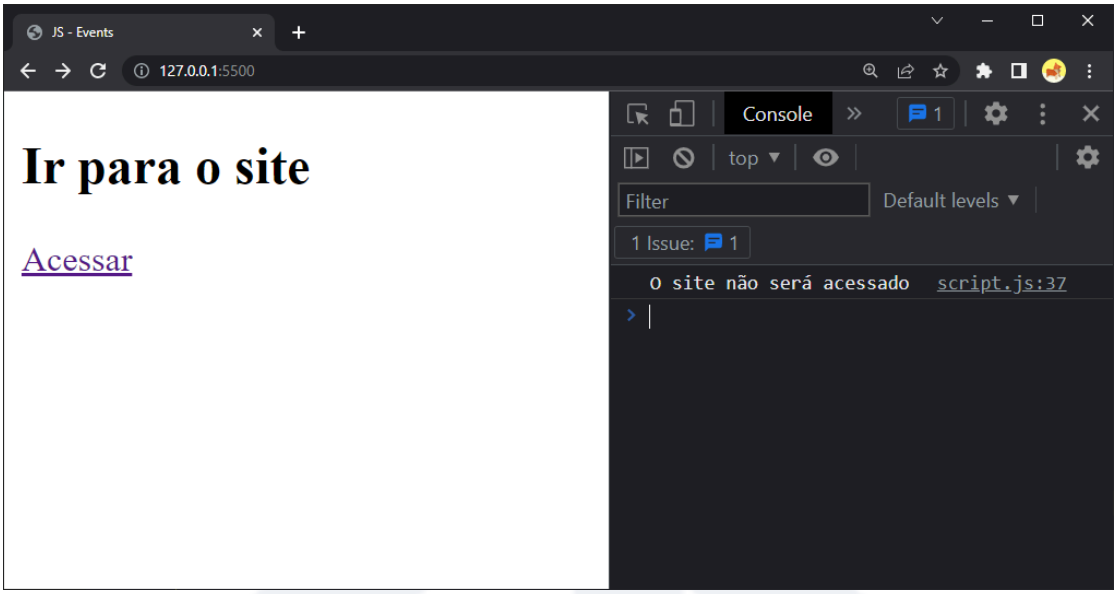

Growdev - Formação completa d
growdev.com.br
growdev
VEM SER O PROFISSIONAL QUE O MERCADO PROCURA E MUDE SUA VIDA.
Quero mudar de vida
Aceleramos sua
Console
top
Filter
Default levels
1 Issue: 1
Console was cleared VM165:1
undefined

O resultado do código acima, de fato, direcionou para o site da Growdev. Porém, repare que antes do acesso ao site a mensagem "O site não será acessado" foi mostrada no console. Isto ocorre por que tanto o comportamento padrão da tag `<a>` quanto o event listener são executados.

Para resolver este problema, o método `preventDefault` – encontrado dentro do objeto de argumento de evento – pode ser usado. Como seu nome sugere, este método **previne o comportamento padrão** de um componente. Desta forma, podemos tomar apenas a ação que definirmos no event listener.

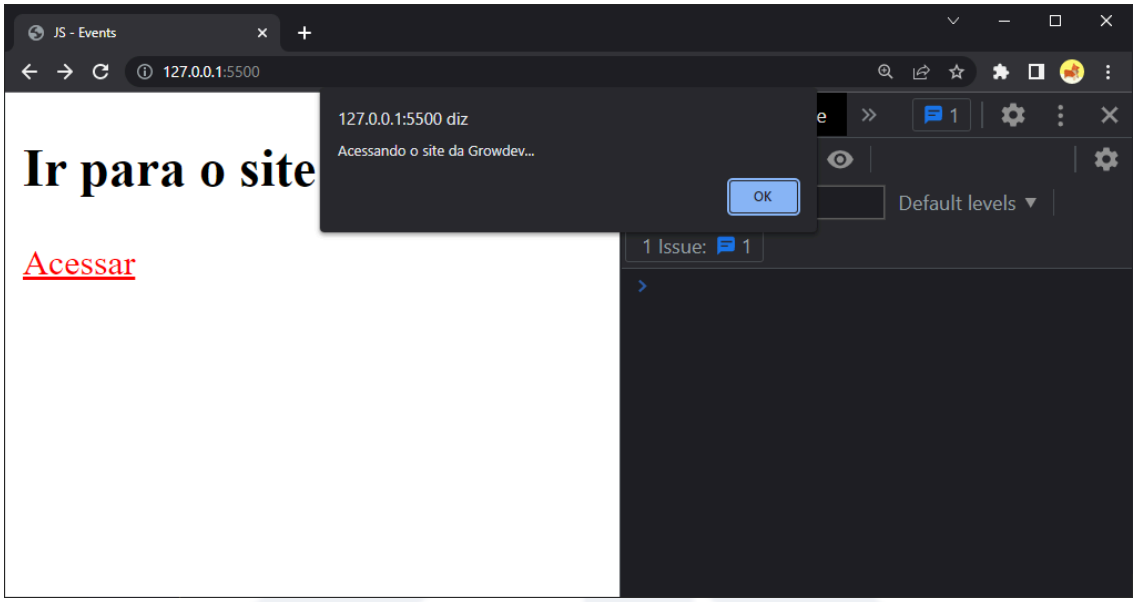
Para usar o método, devemos passar o argumento de evento na função que estamos executando para o evento. Geralmente o método é chamado na primeira linha da função, para que se garanta o impedimento de qualquer comportamento indesejado.

```
const link = document.querySelector("#linkGrowdev");
const someCondition = false;

link.addEventListener("click", (event) => {
  // Previne o comportamento default do link
  event.preventDefault();

  if (someCondition) {
    alert("Acessando o site da Growdev ... ");
    window.location.assign("https://www.growdev.com.br");
  } else {
    console.log("O site não será acessado");
  }
});
```



Neste caso, o resultado do código foi apenas a mensagem no console mas nenhum direcionamento é feito, já que a condição é `false`. Com o `preventDefault` chamado na função, a tag `<a>` deixa de fazer o direcionamento padrão.

Por outro lado, se mudarmos a condição para `true` o direcionamento será feito mas exclusivamente porque o código do listener assim o orienta. Diante desta mudança de condição, a página emitirá um `alert` antes de acessar a página da Growdev.



REFERÊNCIAS

1. Introdução a eventos - Aprendendo desenvolvimento web | MDN
2. Trabalhando com eventos em JavaScript
3. Eventos JavaScript | desenvolvimento para web
4. Eventos em Javascript
5. Eventos em Javascript: Tratando eventos
6. Event reference | MDN
7. Event handling (overview) - Event reference | MDN
8. Working with Inline Event Handlers | HTML Goodies
9. Event.preventDefault() - APIs da Web | MDN
10. event.preventDefault o que é e para que serve - Edson Emiliano
11. Diferença entre event.preventDefault() e event.stopPropagation() | JS para impacientes
12. Javascript Events — The Basics. The interaction of JavaScript with HTML... | by Ramzy Badawy | Medium
13. JavaScript Events Explained

BONS ESTUDOS

growdev