

A large yellow square containing the letters 'JS' in a bold, black, sans-serif font, centered within the square.

JS

eventos

THIAGO DELGADO PINTO
thiago_dp (at) yahoo (dot) com (dot) br

versão: 2020.09.04

eventos são uma forma comum de **notificar** a ocorrência de algo

o navegador provê modelos orientados a eventos
DOM (ex. "*click*"), CSSOM (ex. "*transitionend*")

podemos **registrar tratadores de eventos** e até **criar eventos** personalizados

exemplo

```
<html>
<body>
  <script>
    function ok() {
      alert( 'OK foi clicado.' );
    }
  </script>
  <button onclick="ok()" >OK</button>
</body>
</html>
```

pelo **dispositivo** em que o navegador está rodando

ex. ao mudar a orientação vertical/horizontal, ao mudar suas coordenadas (x, y e z), etc.

pela **janela do navegador**

ex. ao maximizar, ao mudar tamanho, etc.

pelo **processo responsável pela página**

ex. ao montar o DOM, ao carregar completamente, etc.

pela **interação do usuário**

ex. ao pressionar tecla, ao clicar, etc.

pela modificação da estrutura da página (DOM)

por **eventos de mídia**

ex. ao tocar um áudio ou vídeo, ao mudar o volume, etc.

por **eventos de rede**

ex. ao receber a resposta de uma requisição, etc.

etc.

há [várias outras APIs](#) e há como criar eventos personalizados

<https://developer.mozilla.org/pt-BR/docs/Web/Events>

são 28+ páginas A4 de eventos...

eventos comuns

Evento	Ocorre quando...
<code>click</code>	elemento é clicado
<code>load</code>	recurso é carregado
<code>focus</code>	elemento recebe o foco
<code>blur</code>	elemento perde o foco
<code>submit</code>	formulário tem botão de <i>submit</i> pressionado
<code>reset</code>	formulário tem botão de <i>reset</i> pressionado
<code>keydown</code>	tecla é pressionada e navegador ainda não processou a tecla
<code>mouseover</code>	mouse foi movido para um elemento ou um de seus filhos
<code>dragstart</code>	elemento ou texto começa a ser arrastado
<code>error</code>	há erro em um recurso, requisição, evento ou progresso

registro de
eventos

registro de eventos com JavaScript

geralmente via **addEventListener** e **removeEventListener**

agora, por simplificação, vamos considerar que eles têm a seguinte **assinatura**:

```
(type: string, listener: EventListener): void;
```

type é o tipo do evento

listener é a função que irá tratar o evento

registro de eventos – exemplo 1

```
<html>
<body>
  <button id="ok">OK</button>
  <script>
    const aoClicarEmOk = function( event ) {
      console.log( event );
      alert( 'OK foi clicado' );
    };
    document.getElementById( 'ok' )
      .addEventListener( 'click', aoClicarEmOk );
  </script>
</body>
</html>
```

registro de eventos – exemplo 2

```
<html><body>
  <button id="botaoOk" >OK</button>
  <button id="botaoColocar" >Colocar</button>
  <button id="botaoRetirar" >Retirar</button>
  <script>
    const aoClicarEmOk = function() { alert( 'OK foi clicado' ); };
    botaoColocar.addEventListener( 'click', function() {
      botaoOk.addEventListener( 'click', aoClicarEmOk );
      console.log( 'colocou clique' );
    } );
    botaoRetirar.addEventListener( 'click', function() {
      botaoOk.removeEventListener( 'click', aoClicarEmOk );
      console.log( 'retirou clique' );
    } );
  </script>
</body></html>
```

`removeEventListener` requer a **mesma função** passada para `addEventListener`

caso contrário, não remove

registro de eventos – exemplo 3

```
<html><body>
  <button id="botaoA" >A</button>
  <button id="botaoB" >B</button>
  <script>
    const informar = function( event ) {
      alert( event.target.innerText + ' foi clicado' );
    };
    botaoA.addEventListener( 'click', informar );
    botaoB.addEventListener( 'click', informar );
  </script>
</body></html>
```

registro de eventos – exemplo 4

```
<html><body>
  <button id="botaoGrande" >Botão com um texto
grande</button>
  <script>
    const informar = function( event ) {
      console.log( 'Clicado em',
        event.clientX + ',' + event.clientY );
    };
    botaoGrande.addEventListener( 'click', informar );
  </script>
</body></html>
```

registro de eventos – exemplo 5

```
<html><body>
  <input id="nome" type="text" placeholder="Nome" />
  <input id="email" type="email" placeholder="E-mail" />
  <script>
    const informar = function( event ) {
      console.log( 'Saiu de ' + event.target.placeholder );
    };
    nome.addEventListener( 'blur', informar );
    email.addEventListener( 'blur', informar );
  </script>
</body></html>
```

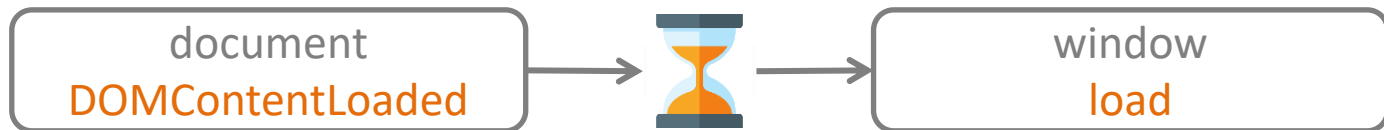
registro de eventos – exemplo 6

```
<html><body>
<script>
  window.addEventListener( 'load', function() {
    console.log( 'Tudo carregado.' );
  } );
  document.addEventListener( 'DOMContentLoaded', function() {
    console.log( 'DOM montado.' );
  } );
  // Imprime "DOM montado" e depois "Tudo carregado"
</script>

</body></html>
```

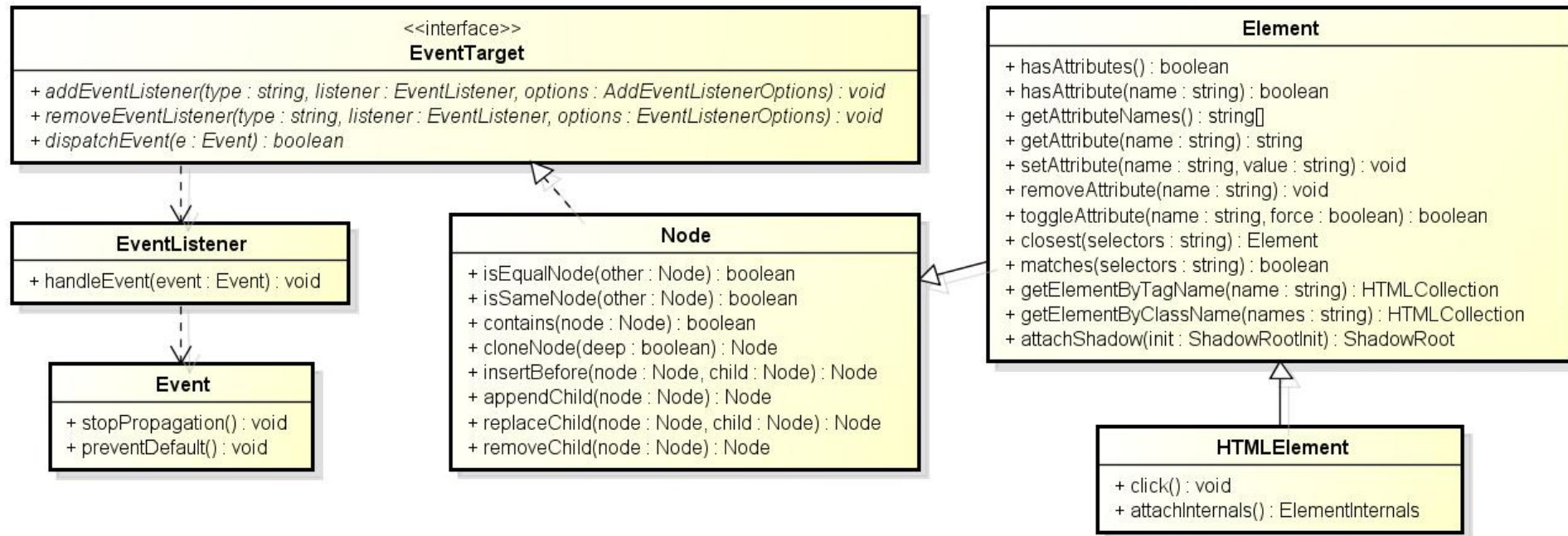

DOMContentLoaded de **document** é disparado quando o navegador **termina de transformar o HTML em DOM**
ocorre **antes** de imagens, frames, CSS, JS e outros recursos serem processados

load de **window** é disparado **após tudo** ser processado



*estrutura e
processamento*

DOM – principais métodos*



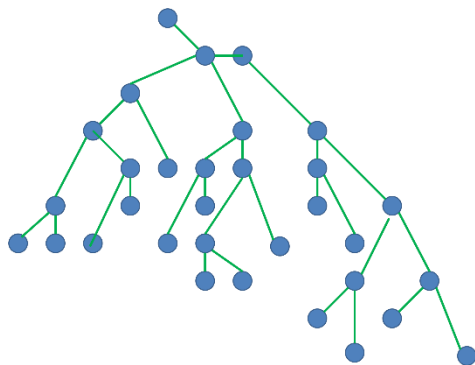
*Com exceção de eventos e métodos relacionados. Algumas interfaces/classes foram omitidas.

disparo de evento

um nó qualquer do DOM tem o método

`dispatchEvent(e: Event): boolean`

esse método dispara um evento na árvore do DOM



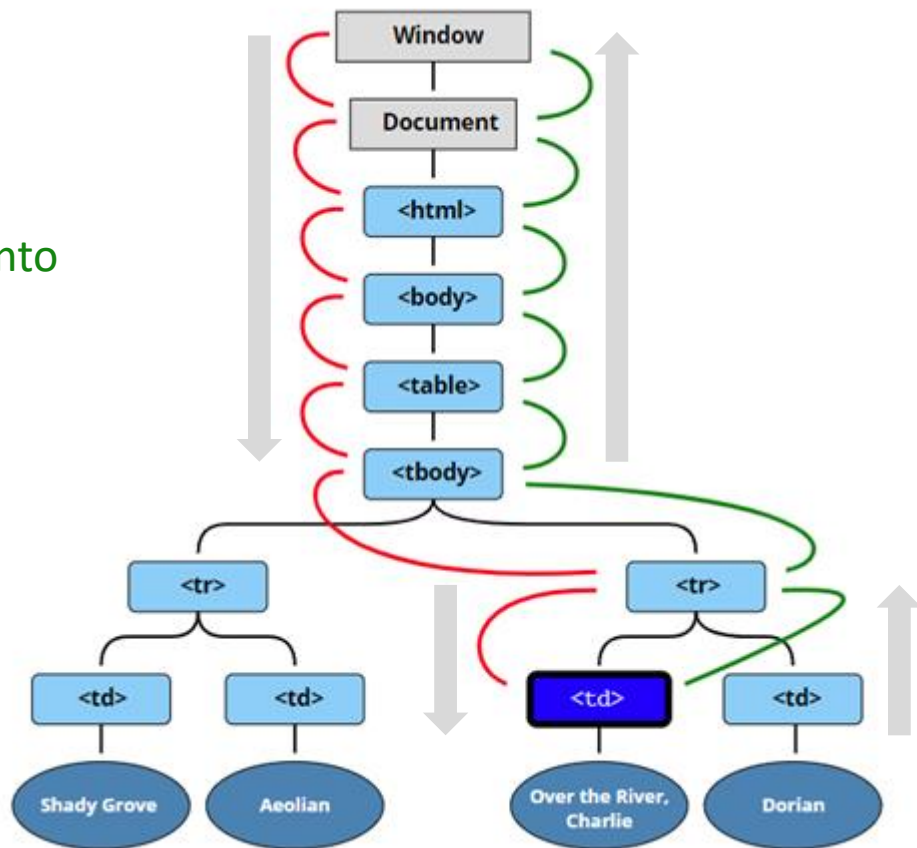
fases do processamento de um evento

1 de 2

1. Fase de Captura

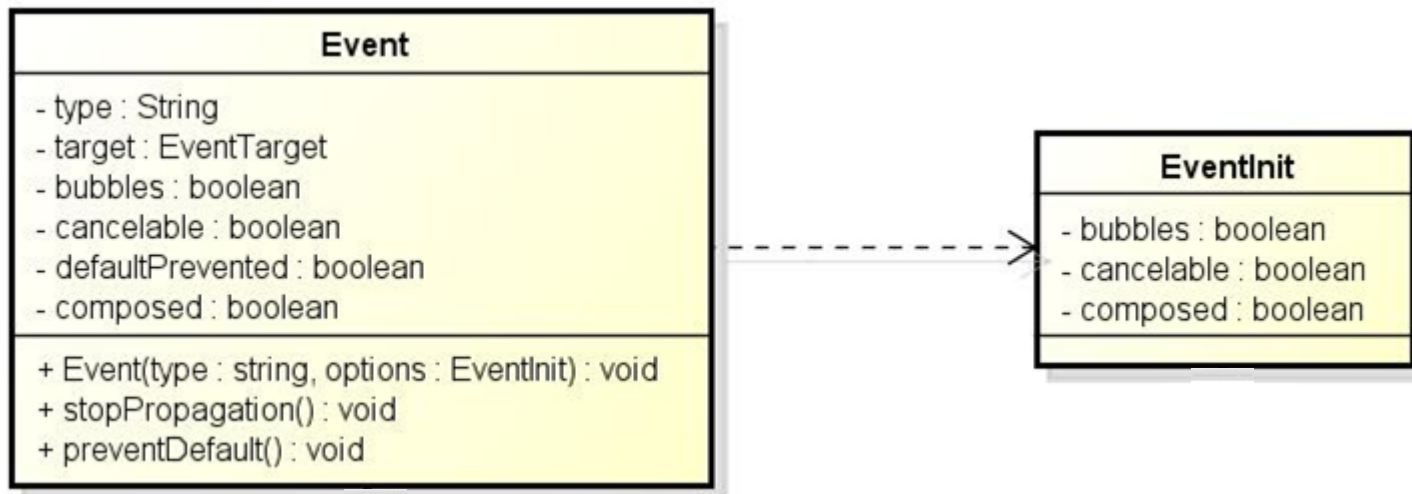
2. Fase do Alvo

3. Fase de Borbulhamento



1. *fase de captura*: o evento propaga de **Window** até o **pai** do elemento alvo
2. *fase do alvo*: o evento chega no **alvo**
3. *fase de borbulhamento*: o evento propaga do **pai** do elemento até **Window**

estrutura de um evento*



*Estrutura parcial. Veja mais em: <https://dom.spec.whatwg.org/#interface-event>

stopPropagation() interrompe a propagação de um evento

ou seja, sua Fase de Borbulhamento

preventDefault() previne que um evento execute seu comportamento padrão – mas não previne a propagação

ex.: por padrão, o clique de um botão colocado dentro de um formulário irá fazer a submissão dos dados desse formulário

exemplo

```
<form>
  <input name="nome" placeholder="Nome" id="nome" />
  <input name="email" placeholder="Email" id="email" />
  <button id="enviar" >Enviar</button>
</form>
<script>
  enviar.onclick = function( ev ) {
    ev.preventDefault(); // Não submete os dados
    ev.stopPropagation(); // Não propaga o clique
    alert( nome.value + ' - ' + email.value );
  };
</script>
```

sintaxe expandida do registro de eventos

de `addEventListener` e `removeEventListener`

ambos têm duas assinaturas possíveis:

```
(type: string, listener: EventListener,  
  options?: {  
    once: boolean = false,  
    passive: boolean = true,  
    capture: boolean = false  
  } ): void;
```

```
(type: string, listener: EventListener,  
  useCapture: boolean = false ): void;
```

sobre options

once indica se é para **remover** automaticamente o *listener* após o evento ocorrer – por padrão é **false**
ou seja, se **true**, o *listener* só executa uma vez

passive indica se o *listener* irá **manter** o comportamento padrão do elemento – por padrão é **true**

capture indica se a captura do evento deve ocorrer na *Fase de Captura* (**true**) ou se na *Fase de Borbulhamento* (**false**)
por default, é **false** (recomendado)

observação sobre a Fase de Captura

seu uso não é tão comum

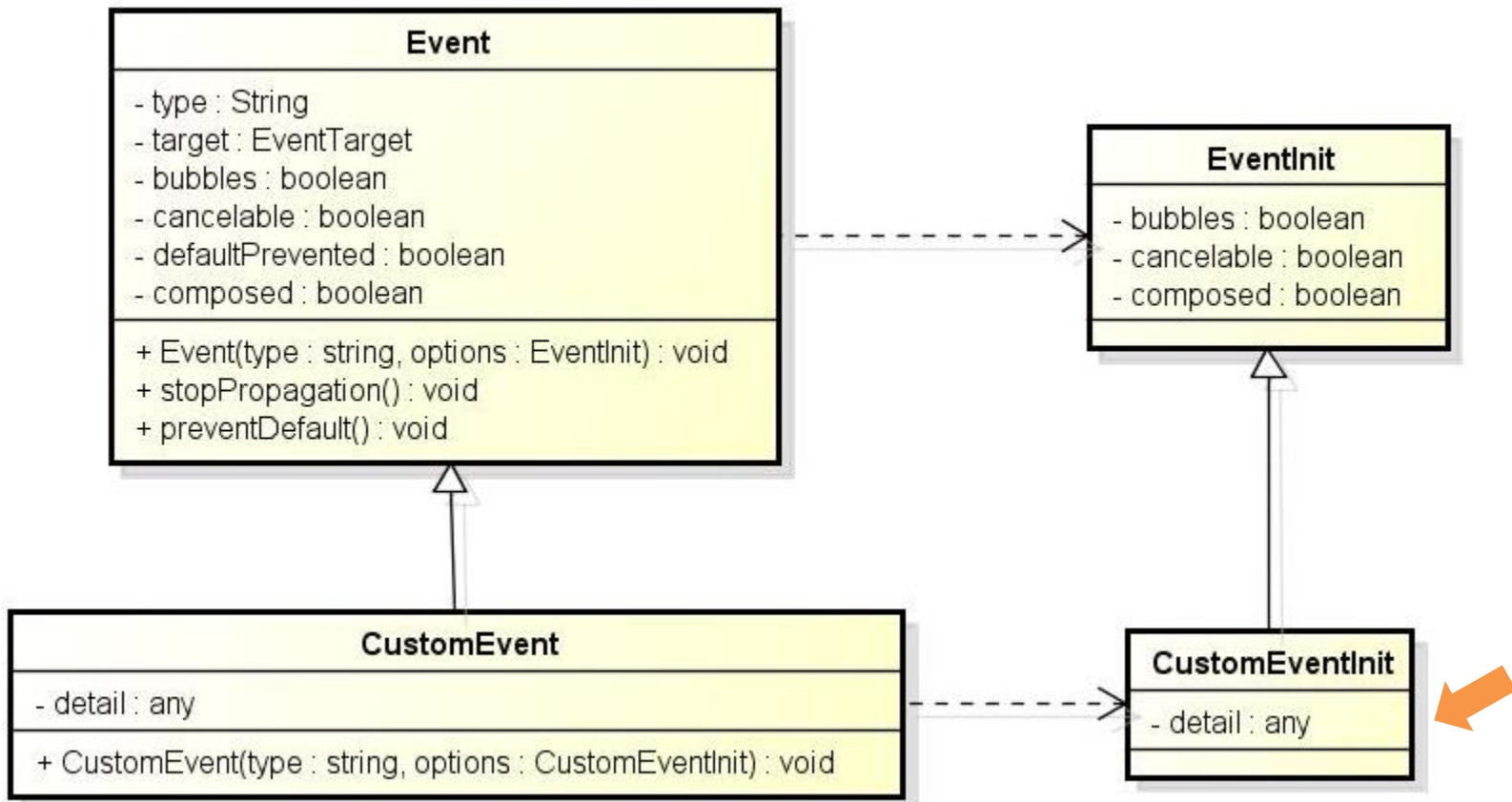
geralmente, só a Fase do Alvo e a de Borbulhamento

um exemplo mais comum é **interceptar** um evento antes de chegar ao alvo

evitar a propagação

*evento
personalizado*

CustomEvent



exemplo

```
<div id="recepcao" >
  <div>Olá,<div id="nome" >Anônimo</div></div>
</div>
<script>
  nome.addEventListener( 'comprimento', function( ev ) { // trata
    nome.innerText = ev.detail.nome;
  } );
  const evento = new CustomEvent( // cria
    'comprimento',
    { detail: { nome: 'Ana' } }
  );
  nome.dispatchEvent( evento ); // dispara
</script>
```

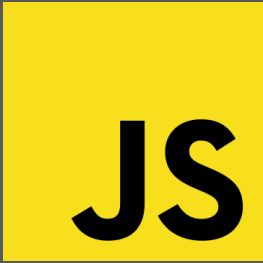
referências usadas

W3C. *UI Events*. Disponível em: <https://www.w3.org/TR/DOM-Level-3-Events/>

MDN. *Event Developers Guide*. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/Guide/Events>

MDN. *Events Reference*. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/Events>

JavaScript.info. *Introduction to Events*. Disponível em: <https://javascript.info/events>



fim

Versão 2020.09.04: Conteúdo inicial.



Licença Creative Commons 4

ESTE MATERIAL PERTENCE AO PROFESSOR THIAGO DELGADO PINTO
E ESTÁ DISPONÍVEL SOB A LICENÇA CREATIVE COMMONS VERSÃO 4.
AO SE BASEAR EM QUALQUER CONTEÚDO DELE, POR FAVOR, CITE-O.