UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

JavaScript na Web

QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

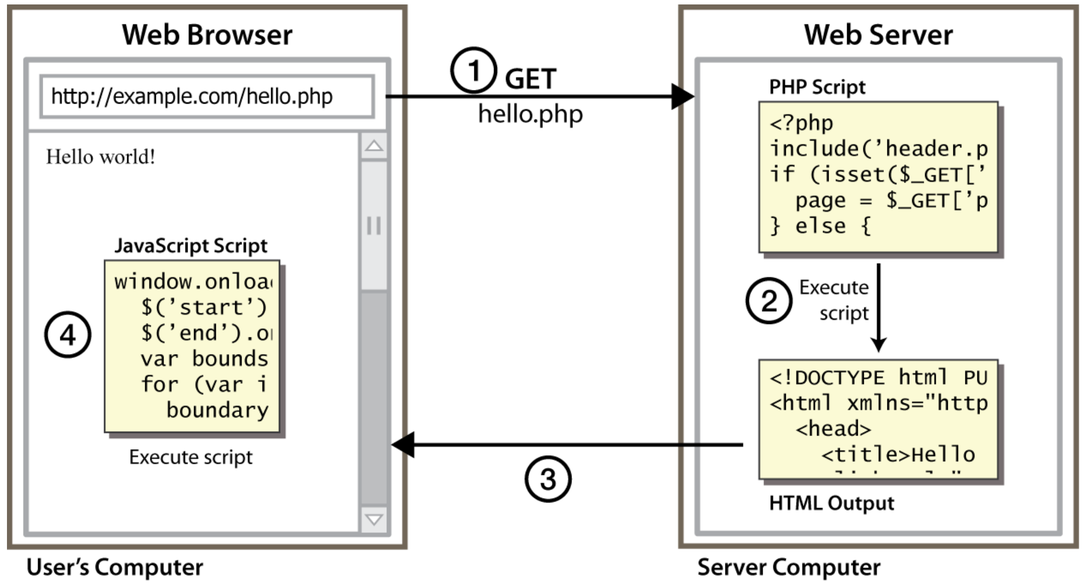
Agenda

- Introdução
- Document Object Model (DOM)
- Browser Object Model (BOM)
- Tratando eventos
- Temporizadores

Introdução

Introdução

- JavaScript pode ser usando tanto no lado do cliente como no lado do servidor
- Focaremos inicialmente no lado do cliente
- Scripts no cliente:
  - Executados no navegador após a página ser enviada de volta pelo servidor
  - Manipulam a página e/ou reagem as ações do usuário



Introdução

Por que usar programação no lado do cliente?

- Benefícios dos scripts no lado do cliente:
  - Usabilidade:
    - Pode modificar uma página sem aguardar por dados do servidor (mais rápido)
  - Eficiência:
    - Pode realizar mudanças pequenas e rápidas sem esperar pelo servidor
  - Orientada a eventos:
    - Pode responder a ações do usuário, como cliques e teclas pressionadas

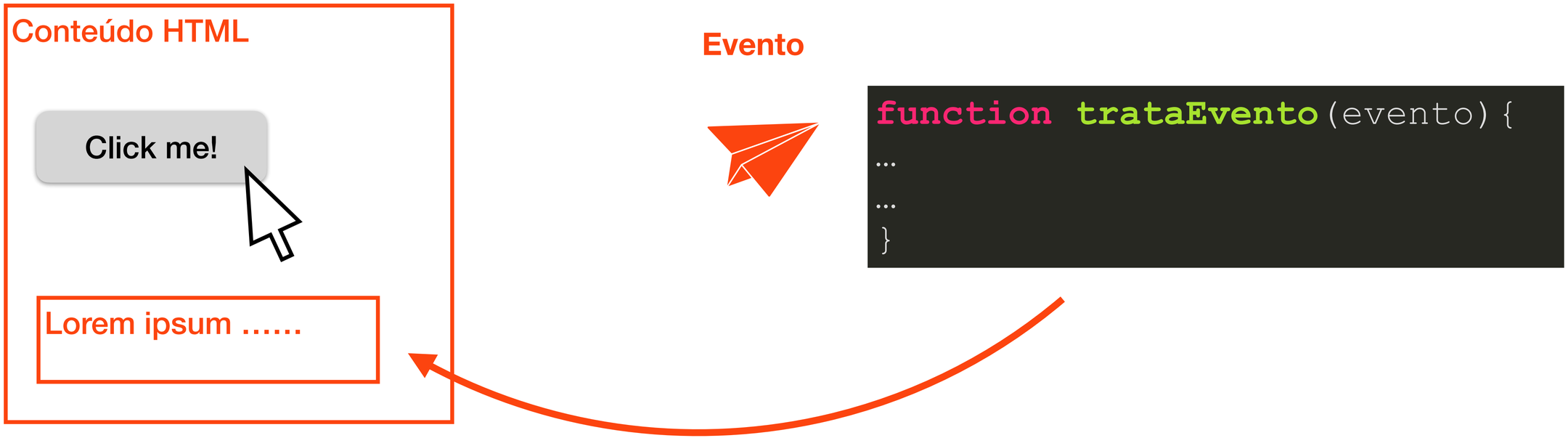
Introdução

- Benefícios de linguagens do lado do servidor:
  - Segurança:
    - Tem acesso a dados privados. O cliente não pode ver o código fonte
  - Compatibilidade:
    - Não depende das implementações dos navegadores
  - Poder:
    - Pode escrever arquivos, abrir conexões com servidores, conectar com banco de dados

Introdução

Programação orientada a eventos

- Programas em JavaScript não possuem uma função principal
- No contexto web eles apenas respondem a ações do usuário que são chamados de eventos



Introdução

Incluindo um javascript

- O código JS pode ser colocado diretamente no arquivo HTML
- No caso de um arquivo externo, a tag script deve ser colocada dentro da tag head
  - O script é armazenado em um arquivo separado com extensão js
  - Atualmente diversos frameworks sugerem por no final da tag body

?

Introdução

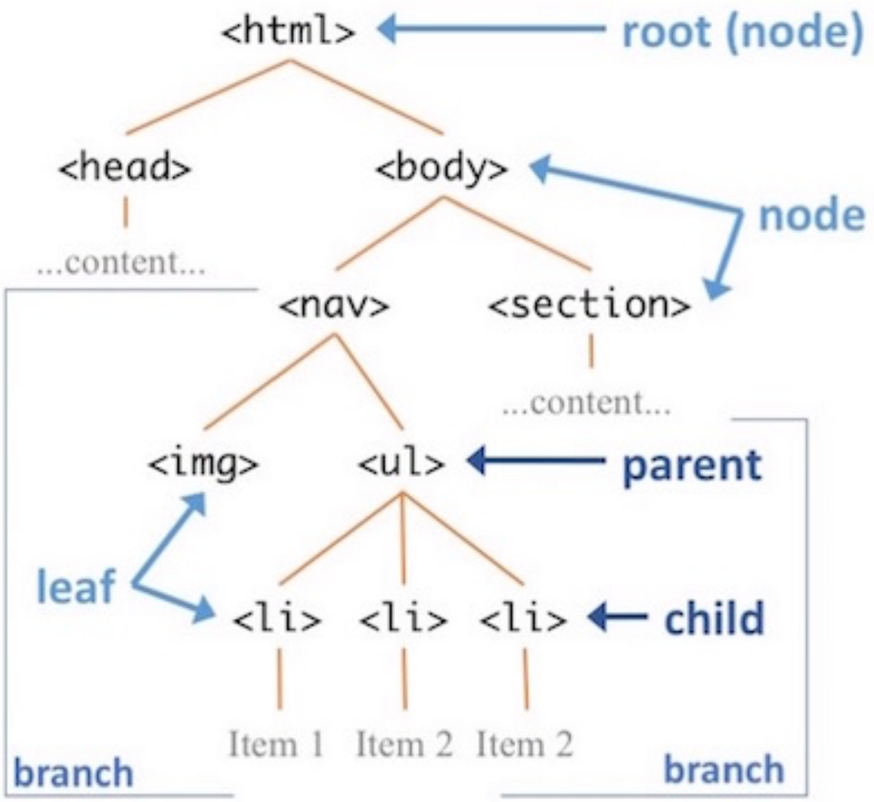
- O código HTML é carregado de forma sequencial
- Logo, por padrão o código js seria carregado antes do HTML
  - O carregamento do código HTML é pausado enquanto script não for totalmente carregado e executado

Verificar compatibilidade: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script#browser_compatibility

Introdução

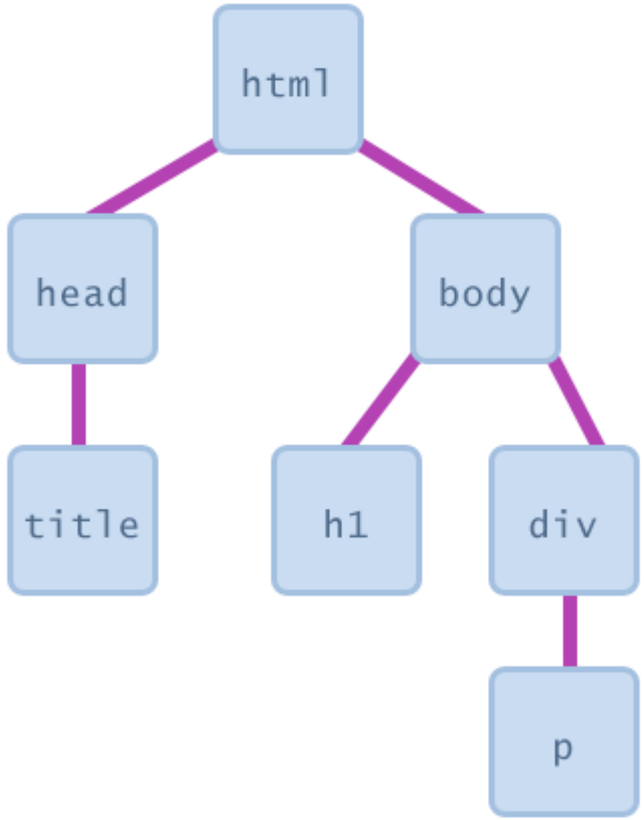
- O tag script está declarada no início ou no final do documento ?
- O script é autônomo ? Ele depende de outro script ou de algum elemento do DOM ?
- É necessário que o DOM esteja totalmente carregado ?

Document Object Model



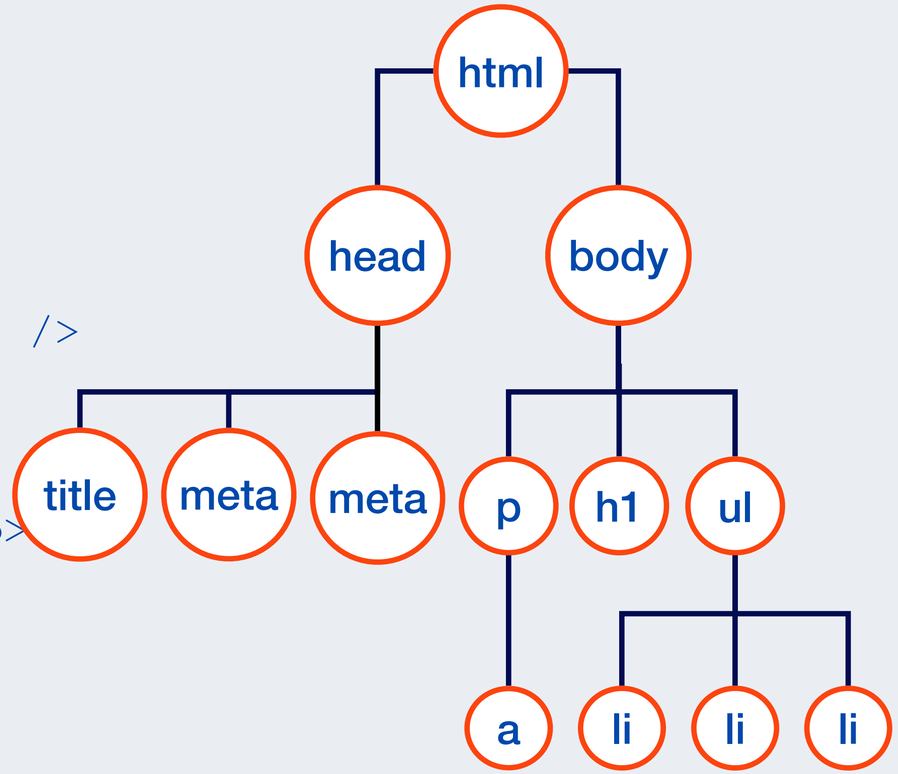
Document Object Model

- A maioria dos códigos JS manipulam uma página HTML
  - Nós podemos verificar o estado de um elemento
    - Ex: Se um checkbox está checado
  - Nós podemos mudar o estado
    - Ex: Inserir um novo texto dentro de uma div
  - Nós podemos simplesmente mudar o estilo
    - Fazer um parágrafo ficar vermelho
- Todos esses elementos possuem um que os presente no DOM



Document Object Model

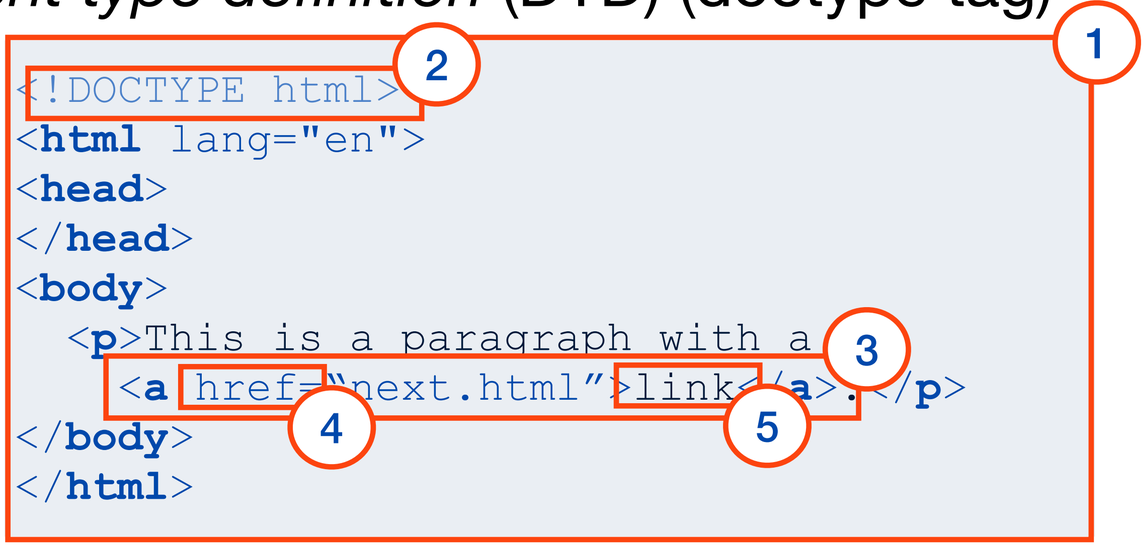
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Sample</title>
  <meta http-equiv="content-type"
   content="text/html; charset=iso-8859-1" />
  <meta http-equiv="Content-Language" content="en-us" />
</head>
<body>
  <p>This is a paragraph of text with a
    <a href="/path/to/another/page.html">link</a>.</p>
  <h1>This is a heading, level 1</h1>
  <ul>
    <li>This is a list item</li>
    <li>This is another</li>
    <li>And another</li>
  </ul>
</body>
</html>
```



Document Object Model

Elemento DOM

- Tipos de nós
  1. Document: Nó raiz de todos os documentos XML - HTML
  2. DocumentType: Representa *document type definition* (DTD) (doctype tag)
  3. Element: Representa uma tag
  4. Attr: Representa um atributo da tag
  5. Text: O conteúdo de um nó
  6. Comment



Document Object Model

Propriedades de um objeto DOM

Vejam todos o métodos e as propriedades:
https://www.w3schools.com/jsref/dom_obj_all.asp

Property / Method	Description
addEventListener()	Attaches an event handler to the specified element
appendChild()	Adds a new child node, to an element, as the last child node
children	Returns a collection of an element's child element
classList	Returns the class name(s) of an element
className	Sets or returns the value of the class attribute of an element
getAttribute()	Returns the specified attribute value of an element node
id	Sets or returns the value of the id attribute of an element
innerHTML	Sets or returns the content of an element
parentNode	Returns the parent node of an element
remove()	Removes the element from the DOM
removeChild()	Removes a child node from an element
setAttribute()	Sets or changes the specified attribute, to the specified value
style	Sets or returns the value of the style attribute of an element

Document Object Model

Acessando elementos

- Existem diversas maneiras de acessar o elemento no DOM
- As mais fáceis são utilizando um dos seguintes métodos
  - document.querySelector(selector): Element
  - document.querySelectorAll(selector): NodeList
  - document.getElementById(id: string): Element
  - document.getElementsByClassName(classname: string): HTMLCollection
  - document.getElementsByTagName(tag: string): HTMLCollection
  - document.getElementsByName(name: string): NodeList

HTMLCollection são atualizadas caso haja mudança no DOM

Document Object Model

Acessando elementos

```
const wrapper = document.querySelector('#wrapper');
wrapper.getElementsByTagName('p');
wrapper.getElementsByClassName('active');
wrapper.getElementsByName(‘something’);

document.querySelector("p.example"); // retorna apenas o primeiro
let x = document.querySelectorAll('#id.class:pseudo'); // retorna todos
```

Document Object Model

Acessando e alterando atributos

- Nós do DOM do tipo Element tem seus atributos expostos
- Podemos acessar essa coleção usando a notação de Array
- Também é possível ler e alterar esses atributos usando os seguintes métodos
  - element.getAttribute('class')
  - element.setAttribute('class', 'new-classname');
  - element.setAttributeNode(attributeNode);
  - element.removeAttribute('class');

Document Object Model

Acessando e alterando atributos

```
<a href="my-awesome-site.html">Clique aqui</a>
<img src="foto.jpg" alt="Esse sou eu!" class="me me-sm" / >
```

```
const image = document.querySelector('.me');
image.src; // returns "foto.jpg"
image.alt = “Essa foto e linda”; // atualiza o texto
image.class; // retorna undefined
image.className; // retorna "me me-sm"

getComputedStyle(image).width

let href = document.getElementById( 'link' ).getAttribute( 'href' );
link.setAttribute( 'href', 'http://www.google.com' );
```

Document Object Model

Acessando e alterando atributos

```
<img src="foto.jpg" alt="Esse sou eu!" class="me me-sm" / >
```

```
const image = document.querySelector('.me');
image.classList; // returns ["me", "me-sm"]
image.classList.add('logo-awesome');
image.classList.remove('me-sm');
image.classList.toggle('active');
image.classList.contains('this-class-doesnt-exist');
image.classList; // now returns [“me", "logo-awesome", "active"]
```

Document Object Model

Ajustando estilos

```
<button id="clickme">Color Me</button>
```

```
let clickMe = document.getElementById("clickme");
clickme.style.color = “red";
clickme.style.backgroundColor = “yellow";

clickMe.style.font-size = "42pt"; // Incorreto
clickMe.style.fontSize = "42pt";

clickMe.style.width = 450; // Incorreto
clickMe.style.width = “450pt";
```

Document Object Model

Boas práticas ao aplicar estilo

Um código JavaScript bem escrito contém o mínimo de código CSS possível

- Use JavaScript para atribuir classes e Id de elementos
- Defina os estilos dessas classes e ids no arquivos CSS

```
clickme.className = "highlighted";
```

```
.highlighted {
  color: red;
  background-color: yellow;
  font-size: 42pt;
  width: 450pt'
}
```

Document Object Model

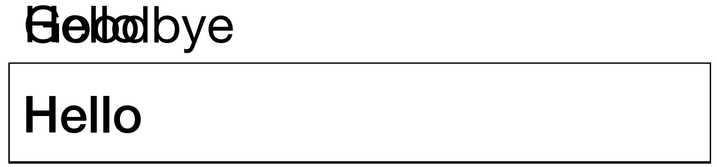
Value vs InnerHTML

- Existem duas maneiras de definir o texto de um elemento, dependendo do seu tipo:
  - innerHTML : texto entre a abertura e fechamento de tags (elementos regulares)
  - value : Elementos parte de formulários
    - Define o valor que será submetido via esse elemento
    - Válido até mesmo para <textarea>

Document Object Model

Value vs InnerHTML

```
<span id="output">Hello</span>
<input id="textbox" type="text" value="Goodbye" />
```




```
function swapText() {
  let span = document.getElementById("output");
  let textBox = document.getElementById("textbox");
  let temp = span.innerHTML;
  span.innerHTML = textBox.value;
  textBox.value = temp;
}

swapText();
```

Document Object Model

Má prática: Uso abusivo de innerHTML

- innerHTML pode ser utilizado para injetar o conteúdo HTML arbitrário na página
  - Tal prática é muito é propensa a erros
  - Torna o código ilegível
  - Procure injetar apenas texto simples

Document Object Model

Alterando a árvore DOM

- É possível alterar a árvore DOM através do Js
- Existem diversas maneiras de criar diferentes de tipos de nós. As mais comuns utilizam os seguintes métodos
  - document.createElement(tag: string): Element
  - document.createAttribute(name: string): Attr
  - document.createTextNode(text: string): Text
  - document.createComment(comment: string): Comment

Document Object Model

Alterando a árvore DOM

- Pare remover elementos em geral utilizamos o seguintes métodos
  - removeChild
  - remove
    - Não funciona no IE

```
const element = document.querySelector('#title');
element.parentNode.removeChild(element);
element.remove(); // this doesn't work on IE!
```

Document Object Model

Alterando a árvore DOM

```
let new_div = document.createElement('div');
let text = document.createTextNode('This is a new div');
new_div.appendChild(text);

let body = document.getElementsByTagName('body')[0];
body.insertBefore(new_div, body.firstChild);
body.replaceChild(new_div, body.firstChild);
div.removeChild(div.firstChild);
```

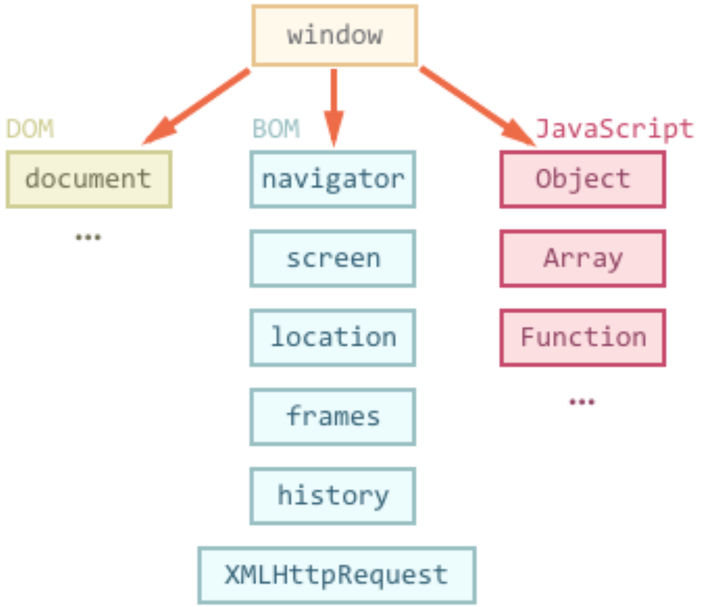
Document Object Model

Alterando a árvore DOM

```
let ul = document.createElement( 'ul' );
document.querySelector('body').appendChild(ul);
let li = document.createElement( 'li' );
li.className = 'check';
for( let i=0; i < 3; i++ ){
  let new_li = li.cloneNode( true );
  new_li.appendChild( document.createTextNode('list item ' + (i + 1)));
  ul.appendChild(new_li);
}
```

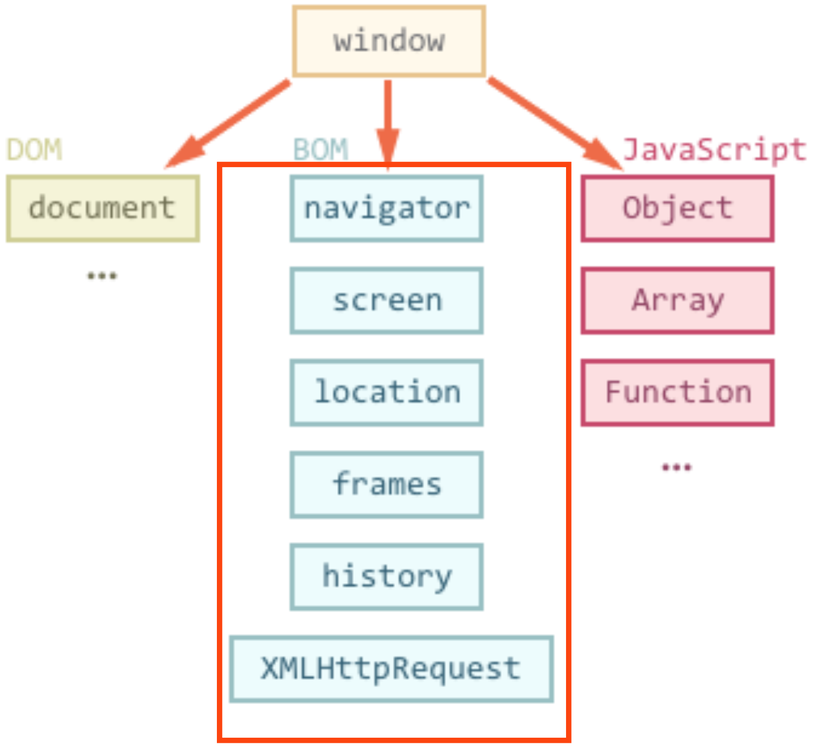
```
<ul>
  <li class="check"> list item 1 </li>
  <li class="check"> list item 2 </li>
  <li class="check"> list item 3 </li>
</ul>
```

Browser Object Model (BOM)



Browser Object Model (BOM)

- Permite a troca de informação não relacionadas ao conteúdo com o navegador
- Apesar de não ser algo padronizado, os navegadores oferecem praticamente as mesmas funcionalidades



Browser Object Model

Objeto	Descrição
window	Raiz da árvore. Todos os outros elementos estão anexados a ele.
history	Fornece informações do histórico do navegador
location	Fornece acesso a informações da URL atual
navigator	Fornece informações sobre o navegador
screen	Fornece informações sobre a área da tela utilizada pelo navegador
frames	Lista de subframe ligados ao objeto window

Browser Object Model

O objeto window

- A janela do browser, o objeto de nível superior na hierarquia DOM
  - Tecnicamente, todo o código global e variáveis são parte do objeto window
- Principais propriedades :
  - document. history, location, name
- Principais métodos:
  - alert
  - setInterval , setTimeout , clearInterval ,clearTimeout (temporizadores)
  - open, close (aparecendo novas janelas do navegador)

Browser Object Model

O objeto history

- A lista de sites no navegador visitou nesta janela
- Propriedades:
  - length
- Métodos:
  - back
  - forward
  - go
- Por motivos de segurança, às vezes o navegador não vai deixar de scripts acessar o histórico

Browser Object Model

O objeto location

- A URL da página web atual
- Propriedades:
  - host
  - hostname
  - href
  - pathname
  - port
  - protocol
  - search
- Métodos:
  - assing
  - reload
  - replace

Browser Object Model

O objeto navigator

- Propriedades:
  - appName
  - appVersion
  - platform
  - language, languages
  - plataform
  - userAgent
  - onLine
  - plugins
- Métodos
  - getBattery( )

Tratando eventos

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Tratando eventos

- Interação do JavaScript com HTML é feita através de eventos que ocorrem quando o usuário ou o navegador manipula uma página
  - Quando a página é carregada, que é um evento
  - Quando o usuário clica em um botão
  - Pressionar qualquer tecla
  - A janela fechar

Tratando eventos

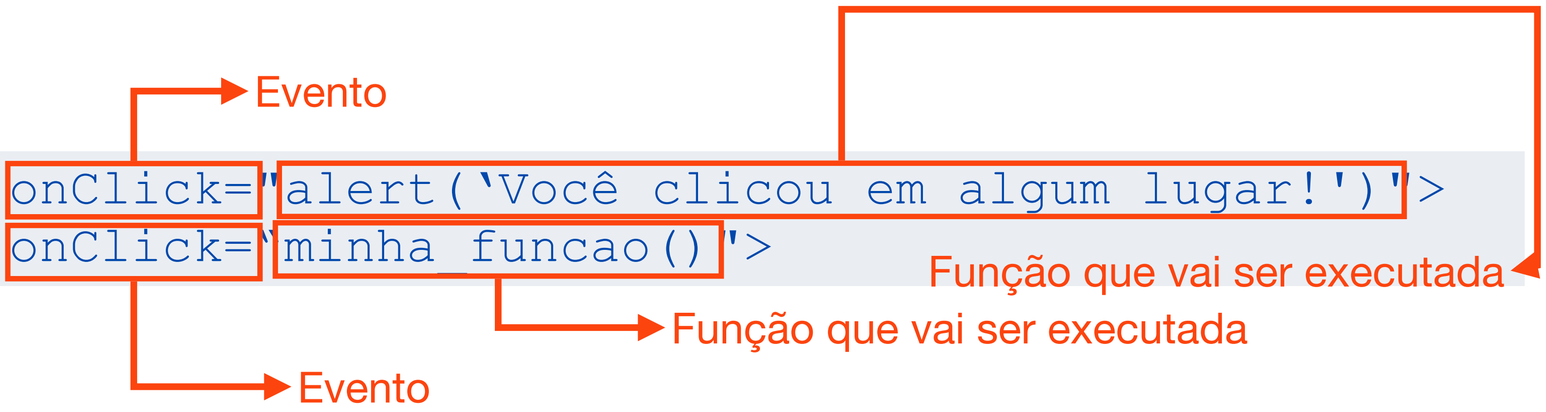
- Podemos usar esses eventos para dar respostas especificas a cada evento
  - Exibir mensagens para os usuários
  - Validar dados
- Cada elemento HTML suporta uma lista própria de eventos
  - É possível escutar e responder a múltiplos eventos de um único elemento
  - Um tipo de evento pode ser gerado por múltiplos elementos

Tratando eventos

- Primeiro passo é registar um handler
- Existem 3 maneiras possíveis
  - Inline
  - Tradicional
  - W3C - Mais recomendada

Tratando eventos

```
<a href="somewhere.html"  onClick="alert('Você clicou em algum lugar!')">
<a href="somewhere.html"  onClick="minha_funcao()">
```



- Maneira mais antiga
- Os tratadores de eventos são atribuídos aos atributos HTML
- Deve ser evitado, o ideal é manter o código javascript totalmente separado do código HTML

Tratando eventos

```
<button id="ok">OK</button>
```

```
let okButton = document.getElementById("ok");
okButton.onclick = minha_funcao;
```

- Suportada incialmente pelo Netscape 3 e IE 4
- É uma boa prática anexar os tratadores de eventos aos objetos dos elementos DOM em seu código JavaScript
- Perceba que você não coloca parênteses após o nome da função

Tratando eventos

```
element.addEventListener('click',startDragDrop,false);
element.addEventListener('click',spyOnUser,false);

element.removeEventListener('click',startDragDrop,false);
```

- Melhor opção
  - Permite vários *handlers* para um mesmo event
  - Ambos os *handlers* serão acionados
    - A ordem não é garantida

Tratando eventos

Eventos de teclado

Evento	Descrição
keydown	Quando o usuário está pressionando um tecla
keypress	Quando o usuário pressiona a tecla
keyup	Quando o usuário libera a tecla

Observatório de eventos do teclado: https://w3c.github.io/uievents/tools/key-event-viewer.html

Ver todos os eventos: https://developer.mozilla.org/en-US/docs/Web/Events

Tratando eventos

Eventos do Mouse

Evento	Descrição
click	Usuário clicou em um elemento HTML
dblclick	Usuário realizou um clique duplo no elemento HTML
mousedown	Quando o usuário pressiona o botão do mouse sobre o elemento HTML
mouseout	Quando o usuário retira o ponteiro do mouse de “cima” elemento HTML
mouseover	Quando o usuário colocar o ponteiro do mouse sobre o elemento HTML
mouseup	Quando o usuário libera o botão do mouse sobre o elemento HTML
mousemove	Quando o mouse é movido enquanto o ponteiro está sobre o elemento HTML

Ver todos os eventos: https://developer.mozilla.org/en-US/docs/Web/Events

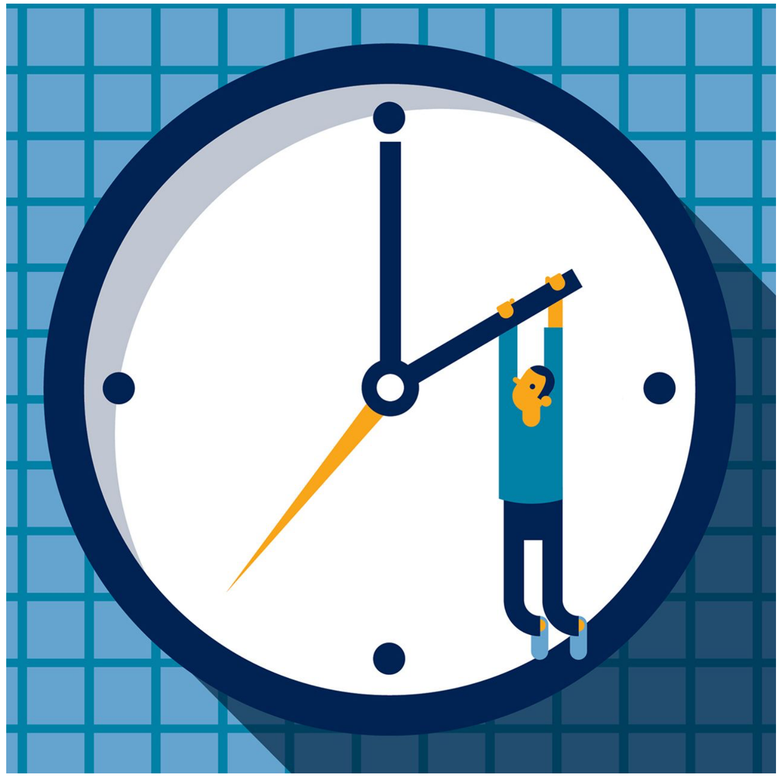
Tratando eventos

Eventos HTML

Evento	Descrição
load	Quando um objeto é carregado
unload	Quando o usuário sai da página
abort	Quando o carregamento de uma media é abortado
error	Quando o ocorre o erro durante o carregamento de um arquivo de media
resize	Quando o document view é redimensionado
change	Quando o conteúdo de um elemento de formulário é alterado
submit	Quando um formulário é submetido
reset	Quando um formulário é resetado
scroll	Quando a scrollbar do elemento é movida
focus	Quando o elemento recebe o foco
blur	Quando o elemento perde o foco

Ver todos os eventos: https://developer.mozilla.org/en-US/docs/Web/Events

Temporizadores



Temporizadores

- O Javascript prover dois mecanismos de tratar eventos relacionados ao tempo
- setTimeout e setInterval retornam um ID que representa o cronômetro
- O ID é utilizado pela função clearTimeout

Método	Descrição
setTimeout	Faz com que uma função seja chamada após o tempo de atraso definido
setInterval	Faz com que uma função seja chamada repetidas vezes a período de tempo
clearTime out, clearInterval	Remove o cronômetro especificado

Temporizadores

setTimeout

```
<button id="myButton">Click me!</button>
<p> Message: <span id="output"></span></p>
```


```
document.getElementById("myButton").addEventListener("click", delayMsg);

function delayMsg() {
  setTimeout(legendary, 3000);
  document.getElementById("output").innerHTML = "Wait for it...";
}

function legendary() {
  document.getElementById("output").innerHTML = "Legendaaarrryyyy!!!";
}
```

Temporizadores

setInterval

```
<button id="myButton">Click me!</button>
<span id="output"></span>
```


```
document.getElementById(“myButton”).addEventListener(“click”, delayMsg);
let timer = null;
function delayMsg2() {
  if (timer == null) { timer = setInterval(chamaMae, 1000);}
   else{
    clearInterval(timer);
    timer = null;
  }
}
function chamaMae() {
  document.getElementById("output").innerHTML += “Mãe!";
}
```

Temporizadores

Passando parâmetros para os cronômetros

```
function delayedMessage() {
  setTimeout(showMessage, 2000, “Oi mãe”, “Outra mensagem” );
}

function showMessage(message) {
  alert(message);
}
```

Referências

- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
- https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript
- http://karloespiritu.github.io/cheatsheets/javascript/
- https://medium.com/@thaisdalencar/no-script-qual-a-finalidade-dos-atributos-async-e-defer-43f2a40533b7
- https://www.w3schools.com/jsref/dom_obj_all.asp
- https://javascript.info/browser-environment
- https://w3c.github.io/uievents/tools/key-event-viewer.html
- https://developer.mozilla.org/en-US/docs/Web/Events
- https://medium.com/@fknussel/dom-bom-revisited-cf6124e2a816

Por hoje é só