



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

- Sem os navegadores possivelmente JavaScript não existiria
- A web nasceu e desenvolveu de forma centralizada
 - Várias empresas desenvolveram seus próprios navegadores
 - Funcionalidades foram desenvolvidas em iniciativas individuais
 - Algumas delas viraram padrões mais tarde

“Por um lado, é empoderador não ter um sistema de controle central, mas tê-la melhorada por várias partes trabalhando em colaboração (ou ocasionalmente hostilidade aberta). Por outro lado, a maneira aleatória pela qual a Web foi desenvolvida significa que o sistema resultante não é exatamente um exemplo brilhante de consistência interna. Algumas partes são francamente confusas e mal concebidas.”

MARIJN HAVERBEKE em *Eloquent JavaScript : a modern introduction to programming*

Introdução

Por que usar programação no lado do cliente?

- Usabilidade e Eficiência
 - Modificações podem ser realizadas sem depender do servidor
- Orientada a eventos
 - Podemos responder a ações do usuário como:
 - Cliques
 - Teclas pressionadas

Introdução

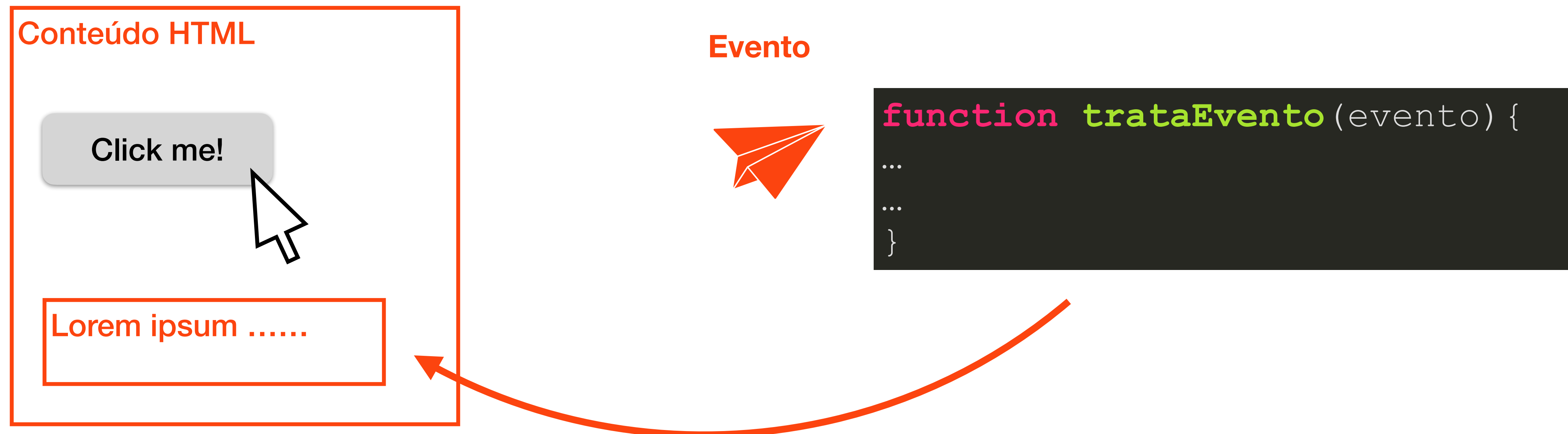
Limitações do uso de linguagens cliente

- Segurança
 - O código é visível para qualquer usuário do cliente
- Compatibilidade
 - Clientes diferentes podem ter implementações diferentes
- Poder computacional
 - Depende do cliente

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
- A **tag <script>** deve ser colocada dentro da **tag <head>**
 - Podemos escrever código HTML diretamente entre **<script></script>**
 - Também é possível utilizar um arquivo externo, em geral com a extensão **.js**
 - **Atualmente, scripts são servidos por alguma CDN**

Introdução

Incluindo um Javascript

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Exemplo</title>
  <meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
  <meta http-equiv="Content-Language" content="pt-br" />
  <script src="exemplo.js"></script>
</head>
<body>
  ...
</body>
</html>
```



- No entanto, na atualidade vários frameworks sugerem que a tag `<script>` seja adicionada **imediatamente antes do fechamentos da tag `<body>`**



Introdução

Linha do tempo de JavaScript do lado do cliente

- 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



Introdução

Linha do tempo de JavaScript do lado do cliente

1

- O navegador cria um objeto **Document** e começa a analisar a página Web, adicionando nós de objetos **Element** e **Text** no documento, à medida que analisa os elementos HTML e seu conteúdo textual. A propriedade **document.readyState** tem o valor “loading” nesse estágio.

2

- Quando o analisador de HTML encontra elementos **<script>** que não têm os atributos **async** ou **defer**, ele adiciona esses elementos no documento e, em seguida, executa o script em linha ou externo. Esses scripts são executados de forma síncrona.
- Quando o analisador encontra um elemento **<script>** que tem o atributo **async** configurado, começa a baixar o texto do script e continua a analisar o documento. O script será executado assim que possível, depois de baixado.

3

- Quando o **documento é completamente analisado**, a propriedade **document.readyState** muda para “interactive”

4

- Qualquer script que tiver o atributo **defer** configurado é executado, na ordem em que aparece no documento. Os scripts assíncronos também podem ser executados nesse momento.

Introdução

Linha do tempo de JavaScript do lado do cliente

5

- O navegador dispara um evento **DOMContentLoaded** no objeto **Document**.
- Isso faz a transição da fase de execução de script síncrono para a fase assíncrona da execução do programa dirigida por eventos.

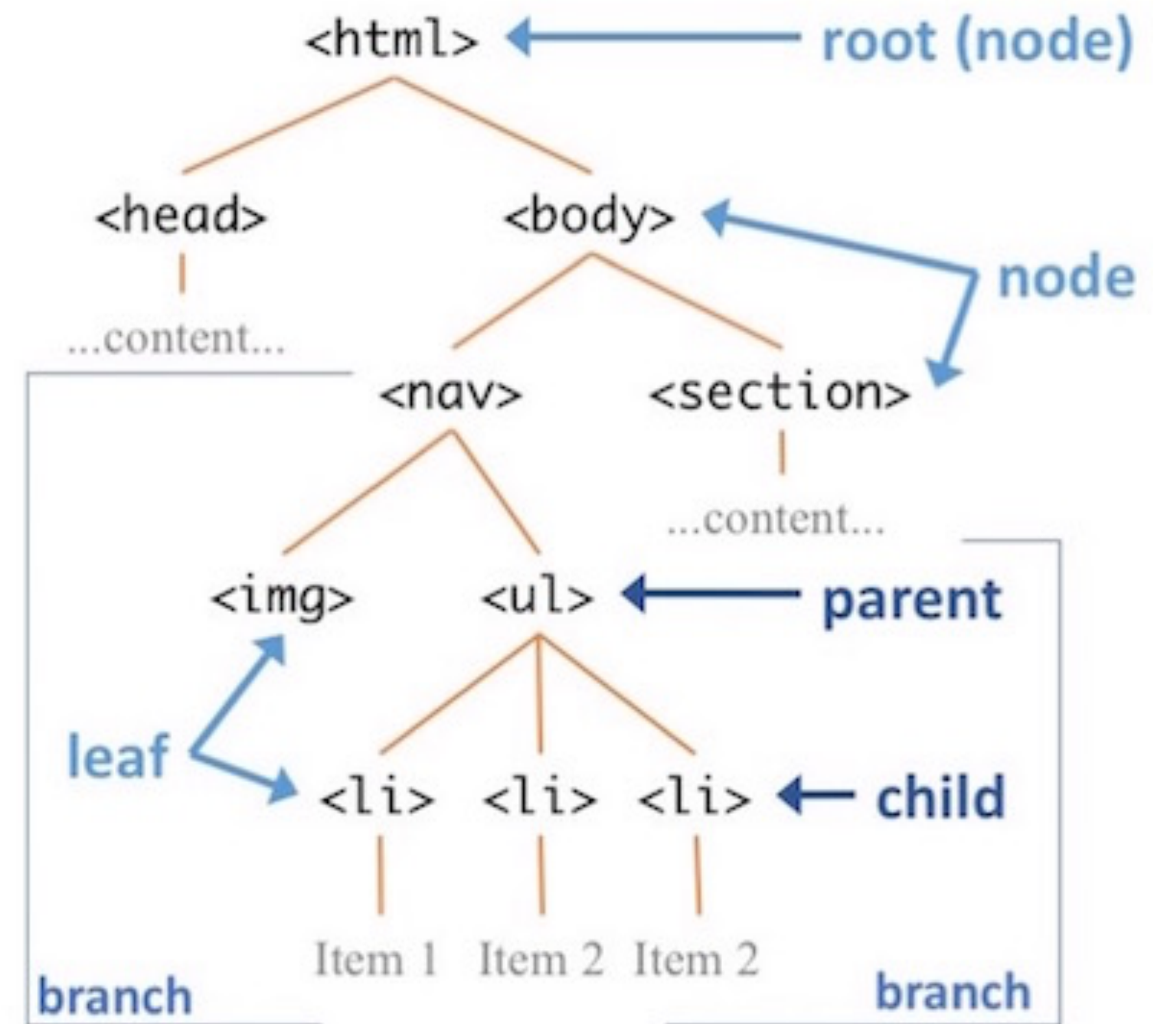
6

- Neste ponto, o **documento está completamente analisado**, mas o navegador **ainda pode estar esperando o carregamento de conteúdo adicional, como imagens**.
- Quando o carregamento de todo o conteúdo termina e **quando todos os scripts async foram carregados e executados**, a propriedade **document.readyState** muda para **“complete”** e o navegador Web dispara um evento de carga (**load**) no objeto **Window**.

7

- Daí em diante, as rotinas de tratamento de evento são chamadas de forma assíncrona, em resposta a eventos de entrada do usuário, eventos de rede, expirações de cronômetro, etc

Document Object Model



Document Object Model

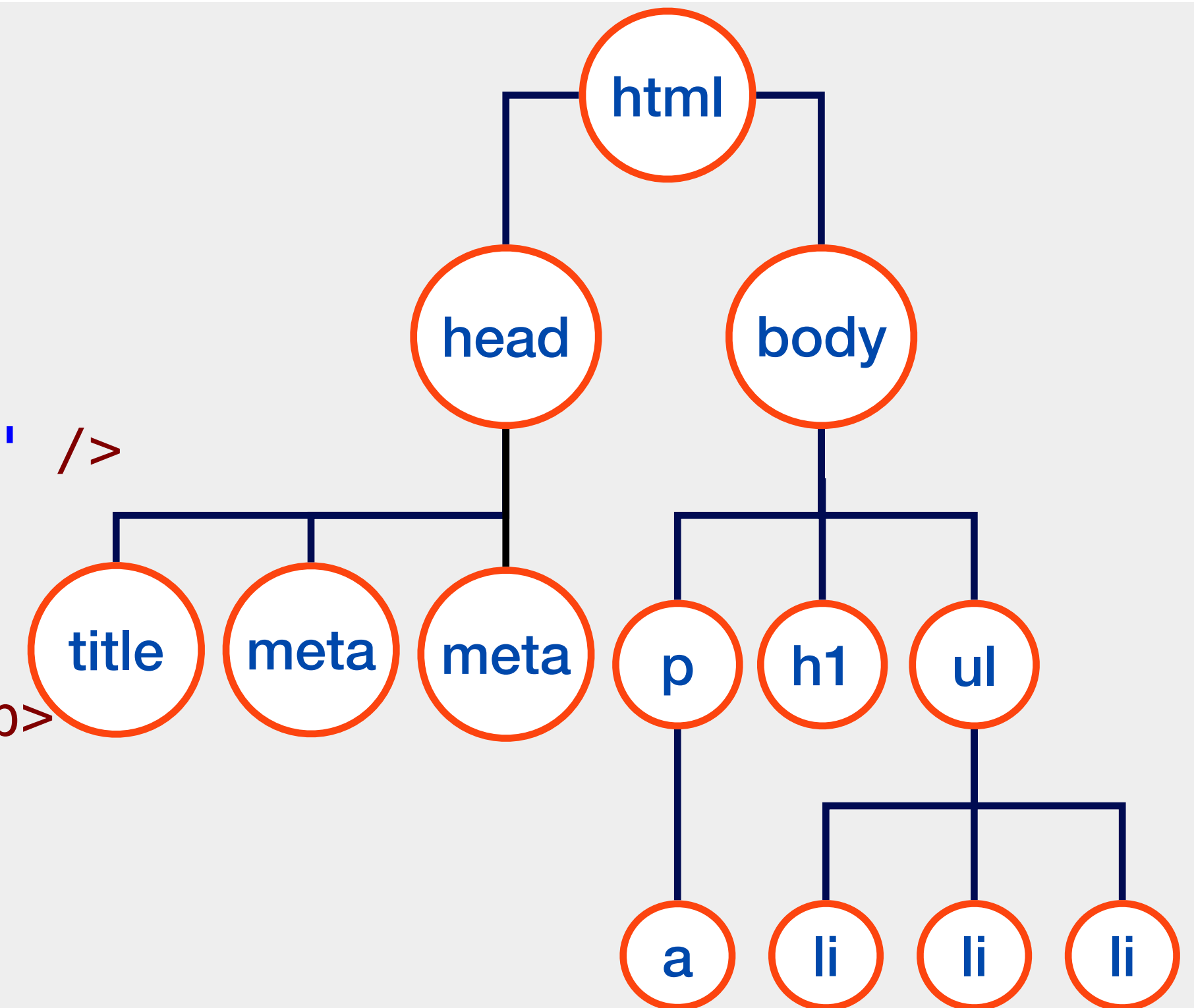
- Quando um usuário abre uma página web no navegador uma sequência de passos é executada:
 1. O navegador carrega o conteúdo text/HTML do arquivo
 2. O navegador realizar o *parser*
 3. O navegador constrói uma modelo estruturado a partir do conteúdo
 4. Usando este modelo, o página web é renderizada

Document Object Model

- O nome desse modelo é: **Document Object Model (DOM)**
 - É possível ler informações nele contidas, assim como podemos alterá-lo
- A maioria dos códigos JS são utilizados para manipular a DOM:
 - Para consultar estados. Ex: se um checkbox está marcado
 - Para alterar a estrutura. Ex: Inserir um novo texto dentro de uma parágrafo
 - Para alterar o estilo de algum elemento. Ex: mudar a cor de texto de um parágrafo

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

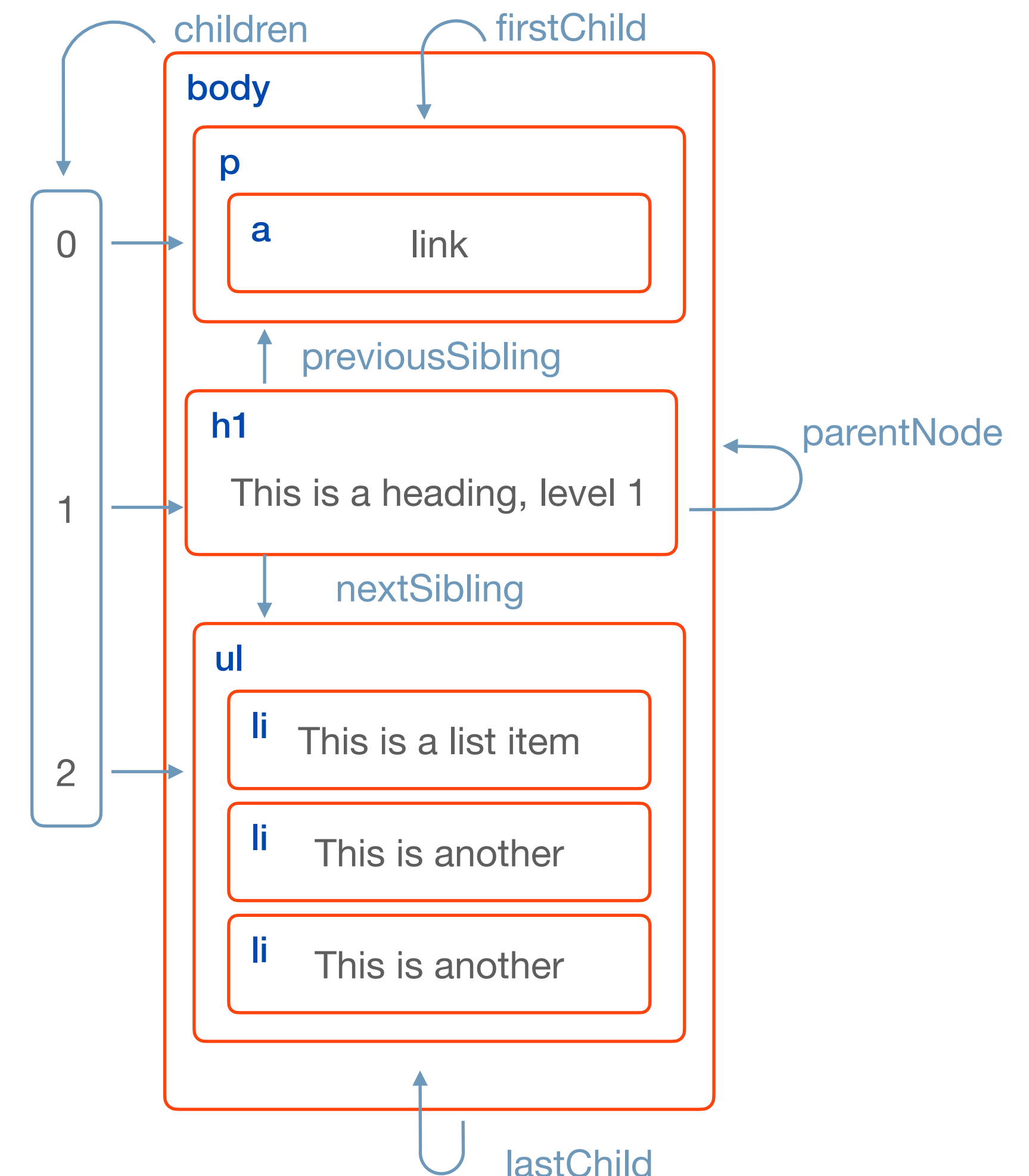
Elemento DOM

- Na DOM podemos encontrar os seguintes tipos de nós (*nodeType*):
 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**

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <p>This is a paragraph with a
    <a href="next.html">link</a>
  </p>
</body>
</html>
```

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

Propriedades de um nó da DOM

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

Propriedade/Métodos	Description
<code>addEventListener()</code>	Anexa um tratador de eventos ao elemento especificado
<code>appendChild()</code>	Adiciona um novo nó filho, a um elemento, como o último nó filho
<code>children</code>	Retorna uma coleção de elementos filhos do elemento
<code>classList</code>	Retorna o(s) nome(s) das classes (CSS) do elemento
<code>className</code>	Define ou retorna o valor do atributo de class do elemento
<code>getAttribute()</code>	Retorna o valor de atributo especificado do elemento
<code>id</code>	Define ou retorna o valor do atributo id do elemento
<code>innerHTML</code>	Define ou retorna o conteúdo do elemento
<code>parentNode</code>	Retorna o nó pai do elemento
<code>remove()</code>	Remove o elemento da DOM
<code>removeChild()</code>	Remove um nó filho de um elemento
<code>setAttribute()</code>	Define ou altera o atributo especificado para o valor especificado
<code>style</code>	Define ou retorna o valor do atributo style de um elemento

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 “*dot notation*”
- 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="meu site.html" id="link">Visite meu site</a>
```

```

```

```
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"
```

dot notation

```
getComputedStyle(image).width
```

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


Document Object Model

O atributo classList

```

```

```
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">Clique em mim</button>
```

```
let clickMe = document.getElementById("clickme");  
clickme.style.color = "red";  
clickme.style.backgroundColor = "yellow";  
clickMe.style.fontSize = "42pt";  
clickMe.style.fontSize = "42pt";  
clickMe.style.width = 450;  
clickMe.style.width = "450pt";
```

Clique em mim

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">Oi</span>  
<input id="textbox" type="text" value="Tchau" />
```

Tchau

Oi

```
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 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

- Para remover elementos em geral utilizamos o seguintes métodos
 - `removeChild`
 - `remove`

Document Object Model

Alterando a árvore DOM

```
let ul = document.createElement( 'ul' );  
document.querySelector( 'body' ).appendChild( ul );
```

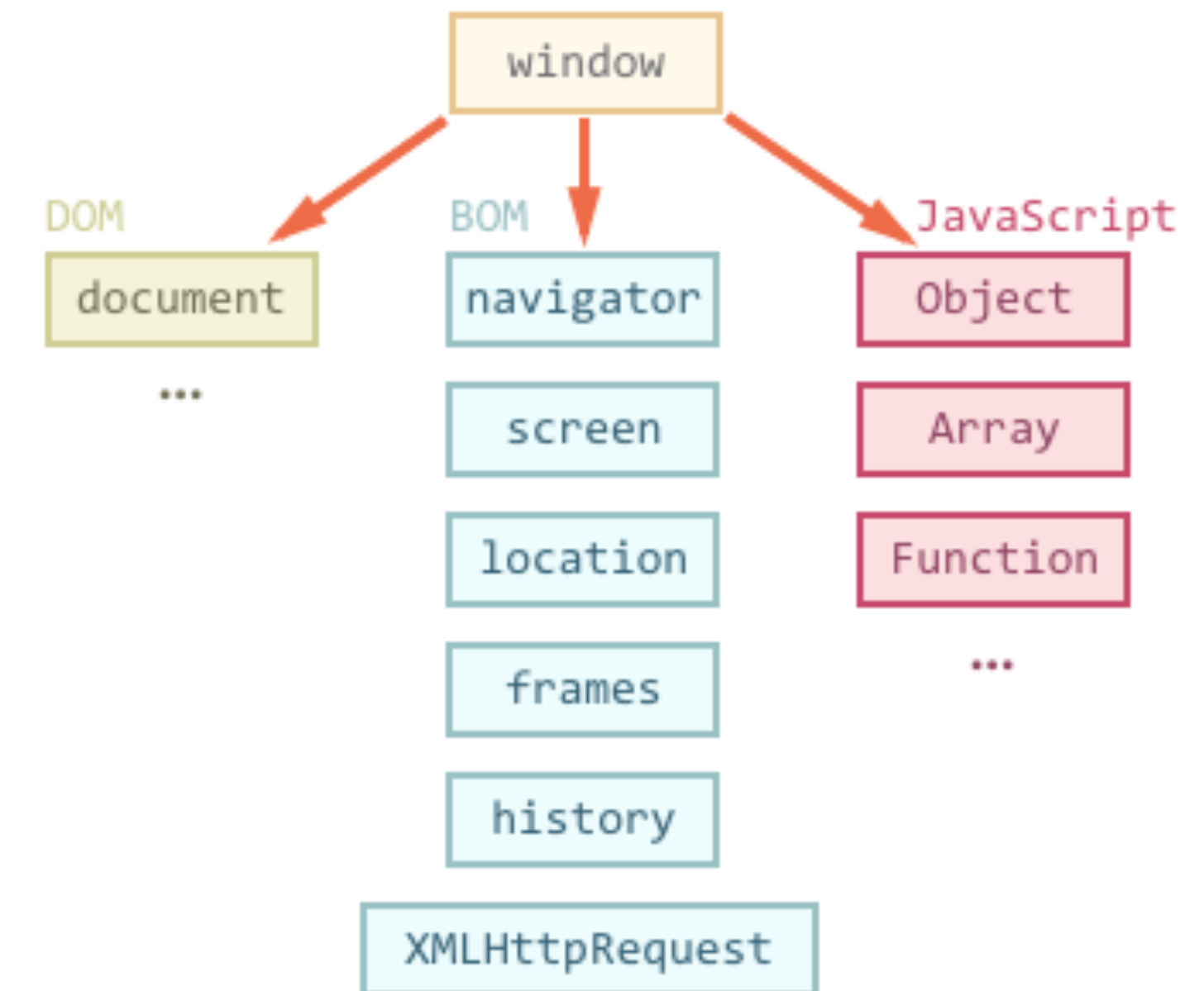
```
let li = document.createElement( 'li' );  
for( let i=0; i < 3; i++ ){  
  let newLi = li.cloneNode( true );  
  newLi.textContent = `list item ${i + 1}`  
  ul.appendChild( newLi )  
}
```

```
ul.removeChild( ul.lastChild )
```

```
ul.remove()
```

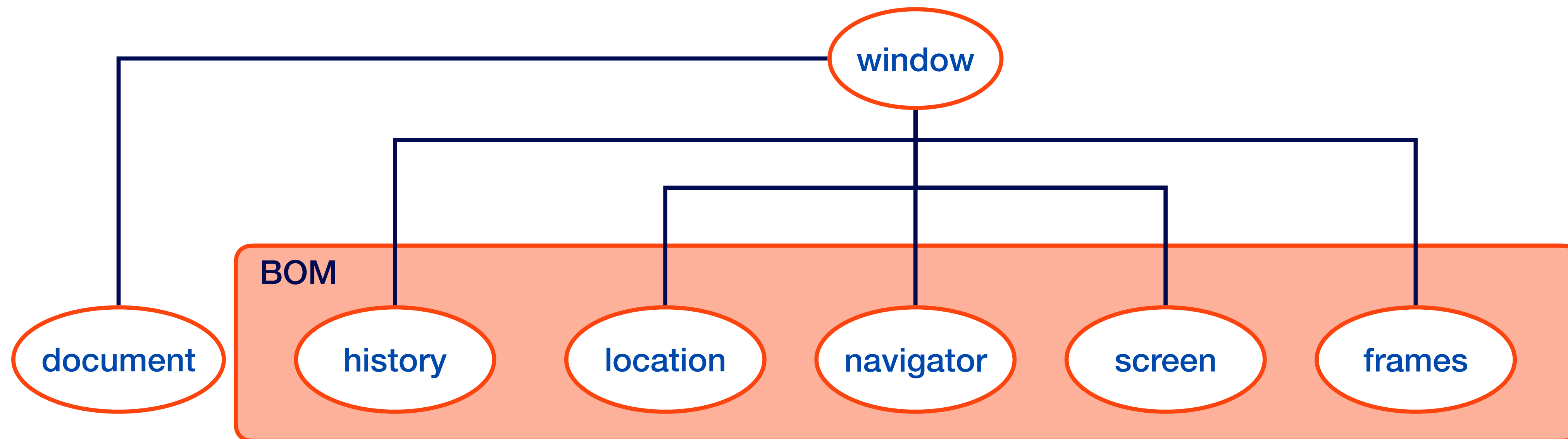
```
<ul>  
  <li class="check"> list item 1 </li>  
  <li class="check"> list item 2 </li>  
  <li class="check"> list item 3 </li>  
</ul>
```

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

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

Propriedades/Métodos	Descrição
<code>history</code>	Fornece informações do histórico do navegador
<code>navigator</code>	Fornece informações sobre o navegador
<code>location</code>	Fornece acesso a informações da URL atual
<code>screen</code>	Fornece informações sobre a área da tela utilizada pelo navegador
<code>alert(string)</code>	Exibe uma mensagem para o usuário e espera que ele a feche
<code>open(url)</code>	Abre uma nova janela no navegador
<code>close()</code>	Fecha a janela atual do navegador
<code>setInterval</code> , <code>setTimeout</code> , <code>clearInterval</code> e <code>clearTimeout</code>	Métodos relacionados a temporizadores

Browser Object Model

O objeto history

- Se refere ao objeto histórico da janela
- Por motivos de segurança, às vezes o navegador não vai deixar de scripts acessar o histórico

Propriedade/Método	Descrição
<code>length</code>	O número de elementos na lista do histórico de navegação
<code>back()</code>	Funcionam como o botão voltar e avançar do navegador
<code>forward()</code>	
<code>go(int)</code>	Pode pular qualquer número de páginas para frente ou para trás na lista do histórico

Browser Object Model

O objeto navigator

- Contém informações sobre o fornecedor e o número da versão do navegador
 - No passado, era em geral usado por scripts para determinar se estavam sendo executados no Internet Explorer ou no Netscape

Propriedades/Métodos	Descrição
<code>userAgent</code>	A string enviada pelo navegador em seu cabeçalho HTTP USER-AGENT
<code>language, languages</code>	Fornece sobre o idioma utilizado no navegador
<code>onLine</code>	Especifica se o navegador está conectado na rede
<code>permissions</code>	Usada para consultar as status de permissão de APIs cobertas pela API Permissions. Apenas leitura
<code>geolocation</code>	Retorna um objeto Geolocation que permite acessar a localização do dispositivo
<code>getBattery()</code>	Utilizado para acessar informações sobre o status de carregamento da bateria
<code>vibrate()</code>	Causa vibração em dispositivos com suporte para isso

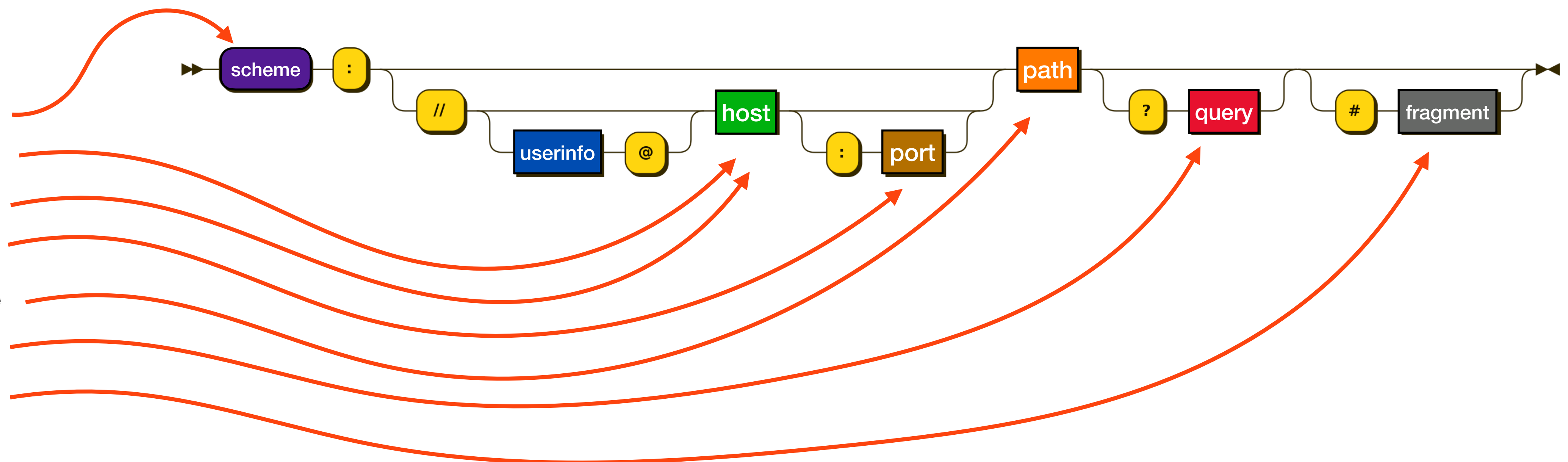
Browser Object Model

O objeto location

- É uma string estática que contém a URL do documento que foi carregado pela primeira vez.

- Propriedades:

- href
- protocol
- host
- hostname
- port
- pathname
- search
- hash




- Métodos:

- assign - faz a janela ser carregada e exibe o documento do URL especificado
- replace - semelhante ao assign, mas remove o documento corrente do histórico de navegação antes de carregar o novo documento
- reload - faz o navegador recarregar o documento.

Manipulando eventos

```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SCRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  };
26 }
```



Manipulando eventos

- Interação do JavaScript com HTML é feita através de eventos que ocorrem quando o usuário ou o navegador manipula uma página. Ex:
 - Quando a página é carregada
 - Quando o usuário clica em um botão
 - Pressionar qualquer tecla
 - A janela vai ser fechada

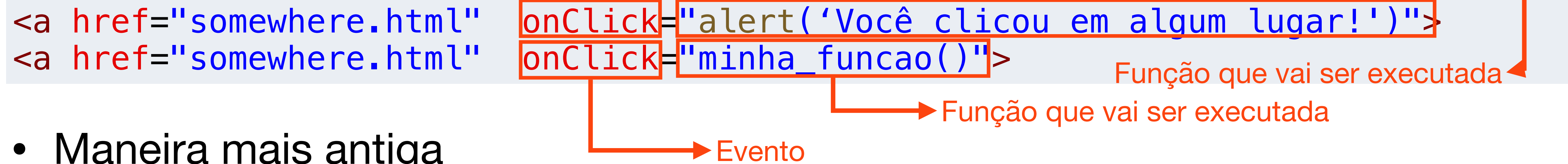
Manipulando eventos

- Podemos usar esses eventos para dar respostas específicas 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

Manipulando eventos

- Primeiro passo é registrar um handler (função “tratadora” de eventos)
- Existem 3 maneiras possíveis
 - Inline
 - Tradicional
 - W3C - Mais recomendada

Manipulando eventos



- 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

Manipulando eventos

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

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

Evento

Função que vai ser executada

- Suportada inicialmente 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

Manipulando eventos

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

Evento ← → Função que vai ser executada

- Permite vários *handlers* para um mesmo evento
 - Ambos os *handlers* serão acionados
 - A ordem não é garantida
- Facilita a remoção individual de um *handler*
- Melhor opção

Manipulando eventos

O objeto *event*

- As funções usadas para tratar eventos sempre recebem como argumento, o objeto *event*
- Este objeto contém informações adicionais sobre o evento
 - É possível renomear o nome do parâmetro

```
let button = document.querySelector("button");
button.addEventListener("mousedown", event => {
  if (event.button == 0) { console.log("Botão da esquerda"); }
  } else if (event.button == 1) { console.log("Botão do Meio"); }
  } else if (event.button == 2) { console.log("Botão da direita"); }
});
```

Manipulando eventos

Propagação

- Para a maioria dos tipos de eventos, as funções “tratadoras” de eventos dos nós pais também serão acionadas quando eventos ocorrerem em nós filhos
- O evento se propaga para fora, do nó onde aconteceu até a raiz do documento
- Ex: Quando um botão dentro de um parágrafo for clicado, as funções “tratadoras” do parágrafo também receberão o evento
 - Os eventos acionam primeiro a função “tratadora” mais específica
- O método *stopPropagation* do objeto de evento pode ser chamado para impedir sua propagação
 - A propriedade de *target* do objeto evento se refere ao nó onde ele se originou

Manipulando eventos

Propagação

```
<p>  
  <button id="myButton">Clique em mim!</button>  
</p>
```

Clique em mim



```
"BUTTON - BUTTON"  
"BUTTON - P"  
"BUTTON - P"
```

```
function showTargets (e) {  
  console.log(` ${e.target.tagName} - ${e.currentTarget.tagName} `)  
}
```

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

```
document.getElementById("myParagraph").addEventListener("click", showTargets);
```

Manipulando eventos

Ações padrões

- Muitos eventos têm uma ação padrão associada a eles
- Exemplo:
 - Se você clicar em um link, será direcionado para o destino do link
 - Se você pressionar a seta para baixo, o navegador rolará a página para baixo.
- Para a maioria dos tipos de eventos, as funções “tratadoras” de eventos são chamados antes que o comportamento padrão ocorra
- Para evitar que o comportamento padrão ocorra podemos chamar o método *preventDefault* no objeto de evento

Manipulando eventos

Eventos de teclado

Evento	Descrição
<code>keydown</code>	Quando o usuário está pressionando um tecla
<code>keypress</code>	Quando o usuário pressiona a tecla
<code>keyup</code>	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>

Manipulando eventos

Eventos de mouse

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

Manipulando 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 pelas funções `clearTimeout` e `clearInterval`

Método	Descrição
<code>setTimeout</code>	Faz com que uma função seja chamada após o tempo de atraso definido
<code>setInterval</code>	Faz com que uma função seja chamada repetidas vezes a período de tempo
<code>clearTimeout, clearInterval</code>	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>  
<p id="output"></p>
```

Click me!

3

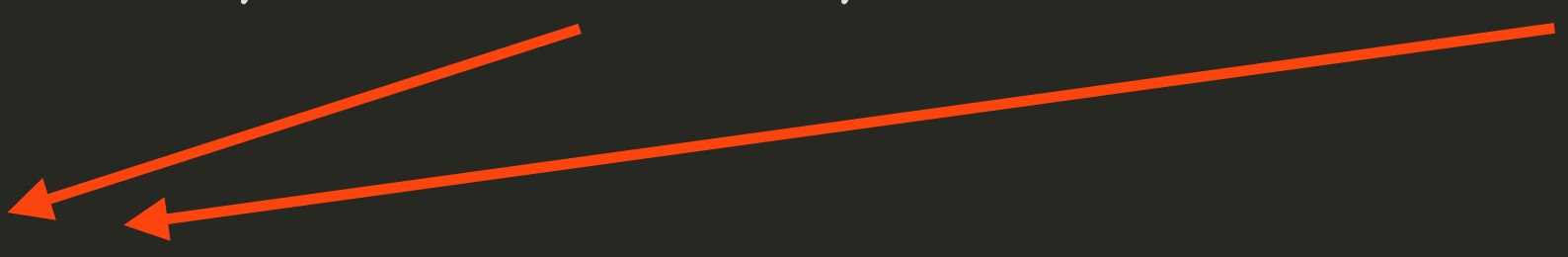
Mãe! Mãe! Mãe!

```
document.getElementById("myButton").addEventListener("click", delayMsg2);  
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