O AMBIENTE DO BROWSER

JavaScript

BOM
DOM

TÓPICOS

- O Browser Object Model (BOM)
- O Document Object Model (DOM)
- Os Eventos do Browser
- O Objecto XMLHttpRequest

INTRODUÇÃO

Browser

PÁGINA DO BROWSER

- O JavaScript numa página do browser tem acesso a variado número de objetos:
  - **Objetos do Core ECMAScript** (todos os mencionados até agora)
  - **Os objetos DOM** (objetos que têm a ver com a página carregada, representada pelo objeto *document*)
  - **Os objetos BOM** (objetos que têm a ver com tudo o que se situa fora da página: janela do browser e ecrã do desktop)
- DOM é um standard mantido pelo W3C (World Wide Web Consortium), existindo diversas versões: DOM Level 1, DOM Level 2, ... (DOM 0 refere-se não é um standard e refere-se ao legado antes do standard);
- BOM não é um standard, embora o HTML5 já inclua alguns dos objetos comuns do BOM.

BOM – BROWSER OBJECT MODEL

JavaScript

O OBJETO WINDOW

- BOM é uma coleção de objetos que permitem aceder ao browser e ao ecrã do computador;
- Tais objetos são acedidos a partir do objeto *window*;
- O objeto *window* define ainda o contexto global de todas as variáveis;

```
> window.variavel = 1;
  1
> variavel;
  1
> parseInt('123a456');
  123
> window.parseInt('123a456');
  123
```

O OBJETO window.navigator

- *window.navigator* é um objeto que permite obter alguma informação sobre o browser:

```
> window.navigator.userAgent;
   "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11;
    rv:42.0) Gecko/20100101 Firefox/42.0"
```

- a propriedade *userAgent* é utilizado para identificar o browser, permitindo implementar diversas versões de código:

```
if(navigator.userAgent.indexOf('MSIE') !== -1)
{
  alert('IE');
}
else {
  alert('Não é IE');
}
```

DETEÇÃO DE CAPACIDADES

- O código anterior não é na realidade o melhor meio para testar o suporte de determinada funcionalidade JavaScript. O melhor é testar se a funcionalidade existe. Exemplo:

```
if(typeof window.addEventListener === 'function') {
  // funcionalidade addEventListener suportada
}
else {
  // funcionalidade addEventListener não suportada.
  // Devemos pensar noutra estratégia
}
```

O OBJECTO window.location

- *Window.location* aponta para um objeto que fornece informação da página atual. Por exemplo, considere que está na página
  `https://www.google.pt/?gfe_rd=cr&ei=_4xlVp-GCOir8wfxw4XwBA&gws_rd=ssl`
  o seguinte código:

```
for(var i in location){
  if(typeof location[i] === 'string') {
      console.log(i + ' = "' + location[i] + '"');
  }
};
```

  mostra um conjunto de informação sobre o url da página atual.

O OBJECTO window.location

- ***Window.location*** **disponbiliza um conjunto de métodos:**

```
// muda de página sem registar no histórico
> location.replace('http://www.istec.pt');

// muda de página registando no histórico
> location.assign('http://www.google.com');

// recarrega a página atual
> location.reload();
```

O OBJETO window.history

- ***window.history*** **permite o acesso às páginas visitadas (acesso ao histórico de de navegação);**

```
// devolve nº de páginas no histórico
> history.length;

// navega nas páginas visitadas
> history.forward();
> history.back();
> history.go(-1);

// recarrega a página atual
> history.go(0);
```

O OBJETO window.screen

- O objeto *window.screen* permite obter informação do ambiente exterior ao browser.

```
// devolve o nível de cor
> screen.colorDepth;

// devolve as dimensões do ecrã total e disponível
> screen.width;
> screen.availWidth;
> screen.height;
> screen.availHeight;
```

OS MÉTODOS window.open() / close()

- O método window.open() permite abrir um nova janela do browser. Este método aceita os seguintes parâmetros: URL, nome da janela, lista de funcionalidade (width, height, resizable, status).

```
// Abre uma nova janela
 > var win = window.open(
                'http://www.istec.pt','ISTEC',
   'width=300, height=300, resizable=yes');

 // fecha a janela aberta anteriormente
 > win.close();
```

OS MÉTODOS window.setTimeout() E window.setInterval()

- **Os métodos window.setTimeout() e window.setInterval() permitem agendar a execução de um pedaço de código:**
- **Exemplos:**

```
// executa o método hello() passado 2seg
> function hello() {console.log('olá!');
> var id = setTimeout(hello, 2000);


// executa o método hello() repetidamente
//de 2 em 2 seg.
> var id = setInterval(hello, 2000);
// para e destrói o agendamento de execução
> clearTimeout(id);
```

DOM - DOCUMENT OBJECT MODEL

JavaScript

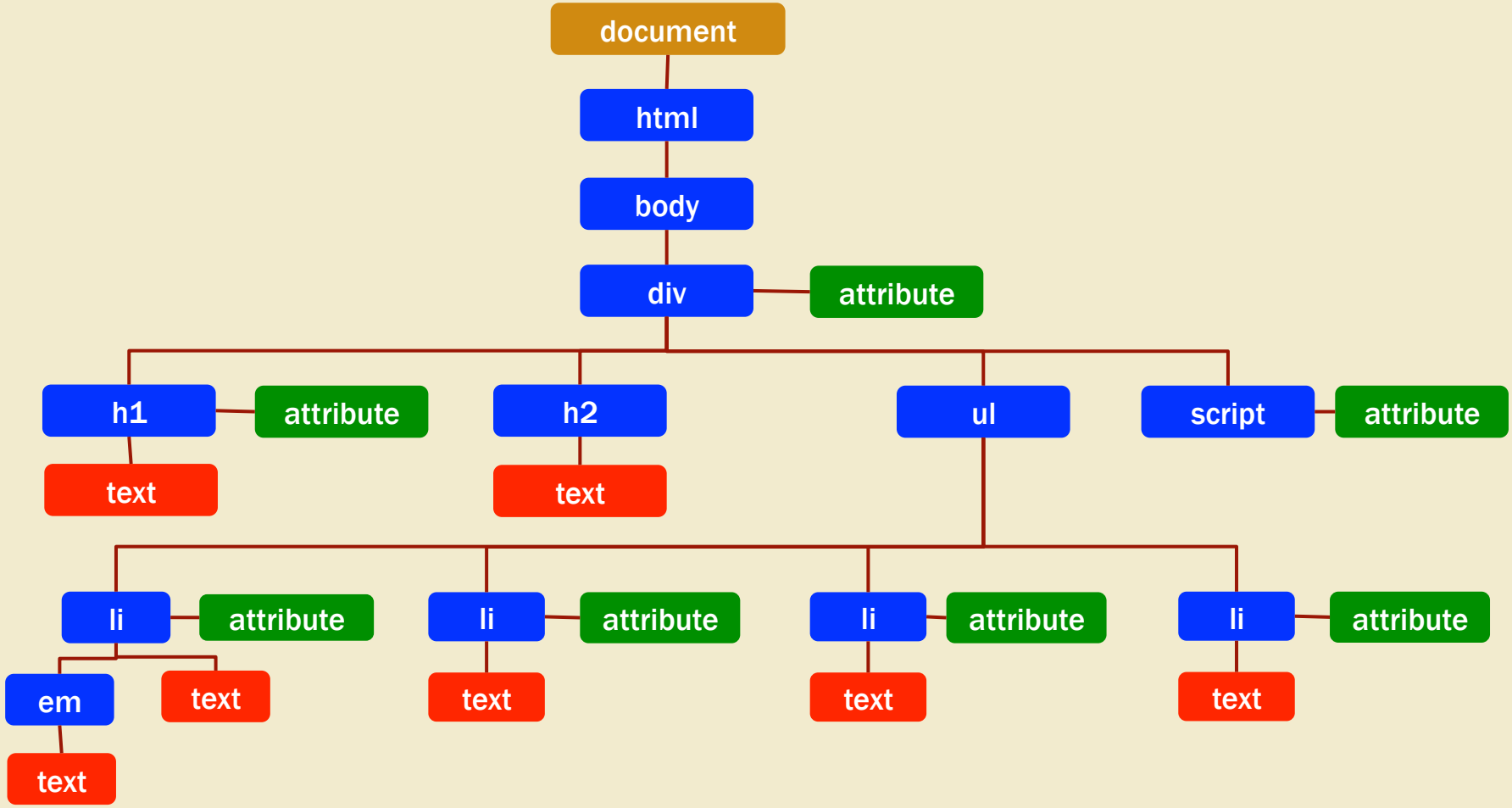
O OBJETO window.document

- O objeto *window.document* guarda toda a informação da página atual.
- Este objeto representa um documento XML ou HTML como uma árvore de nós;
- Os métodos e propriedade de objeto document permite aceder, modificar, remover ou inserir novos elementos na página activa;
- DOM é um API independente que pode ser implementada não apenas em JavaScript mas também noutra linguagem.

A PÁGINA HTML

```
<html
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</
em>figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
   </body>
</html>
```

ÁRVORE DOM DA PÁGINA HTML



ACEDER AOS ELEMENTOS DOM

- Selecionar um elemento (node)
  - getElementById('id')
  - querySelector('seletor css')
- Selecionar vários elementos (nodelist)
  - getElementsByClassName('class')
  - getElementsByTagName('tag')
  - querySelectorAll('seletor css)
- Saltando entre elementos
  - parentNode
  - previousSibling
  - nextSibling
  - firstChild
  - lastChild

MANIPULAR OS ELEMENTOS

- Aceder/atualizar texto dos elementos
  - nodeValue // acede ao texto a partir do nó
- Trabalhar com o conteúdo HTML
  - innerHTML // acede/altera texto e markup
  - textContent // acede/altera texto
  - innerText // acede/altera texto
  - createElement()
  - createTextNode()
  - appendChild()
  - removeChild()
- Aceder ou atualizar os valores dos atributos
  - className // atributo class
  - Id // atributo id
  - hasAttribute()
  - getAttribute()
  - setAttribute()
  - removeAttribute()

DOM EVENT

JavaScript

EVENTOS

USER INTERFACE EVENTS	
Evento	**Descrição**
load	A página web terminou de ser carregada
unload	A página web deixou de estar carregada
error	Browser encontrou um erro de JavaScript ou um dados não existe
resize	A janela do browser foi redimensionada
scroll	O Utilizador efetuou scroll up ou down da página

KEYBOARD EVENTS	
Evento	**Descrição**
keydown	O utilizador pressionou uma tecla
keyup	O Utilizador libertou uma tecla
keypress	Um caratere foi inserido

EVENTOS

MOUSE EVENTS	
Evento	**Descrição**
click	O utilizador pressionou e libertou o botão em cima do mesmo elemento
dblclick	O utilizador pressionou e libertou o botão duas vezes em cima do mesmo elemento
mousedown	O Utilizador pressionou o botão do rato em cima de um elemento
mouseup	O Utilizador libertou o botão do rato em cima de um elemento
mousemove	O Utilizador moveu o rato
mouseover	O Utilizador moveu o ponteiro do rato por cima de um elemento
mouseout	O Utilizador moveu o ponteiro do rato para fora de um elemento

FOCUS EVENTS	
Evento	**Descrição**
focus /focusin	O elemento ganhou o foco
Blur / focusout	O elemento perdeu o foco

EVENTOS

FORM EVENTS	
Evento	**Descrição**
input	O valor num elemento <input> ou <textarea> foi alterado (IE9+) ou qualquer elemento com o atributo *contenteditable*
change	O valor numa *select box*, *checkbox* ou *radio button* foi alterado (IE9+)
submit	O utilizador submeteu o formulário
reset	O utilizador clicou no botão de *reset* do formulário
cut	O utilizador efetuou *cut* do onteúdo de um elemento do formulário
copy	O utilizador efetuou *copy* do conteúdo de um elemento do formulário
paste	O utilizador efetuou *paste* num elemento do formulário
select	O utilizador selecionou texto de um elemento do formulário

EVENTOS

MOTATION EVENTS	
Evento	**Descrição**
DOMSubTreeModified	Uma alteração foi efetuada no objecto *document*
DOMNodeInserted	Um novo elemento foi inserido como um elemento filho de outro elemento
DOMNodeRemoved	Um elemento foi removido de outro elemento
DOMNodeInsertedIntoDocument	Um elemento foi inserido como um descendente de outro elemento
DOMNodeRemovedFromDocument	Um elemento foi removido como descendente de outro elemento

MODOS DE VINCULAR EVENTOS

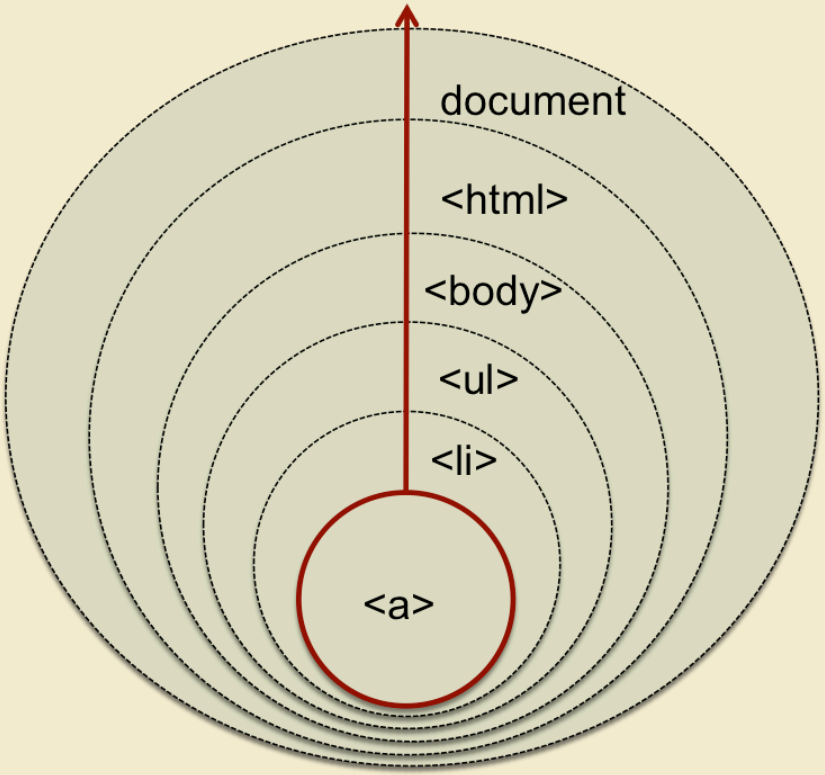
- HTML event handlers (não recomendado)
- DOM event handlers tradicional
- DOM Level 2 event listeners

FLUXO DE ATIVAÇÃO DE EVENTOS

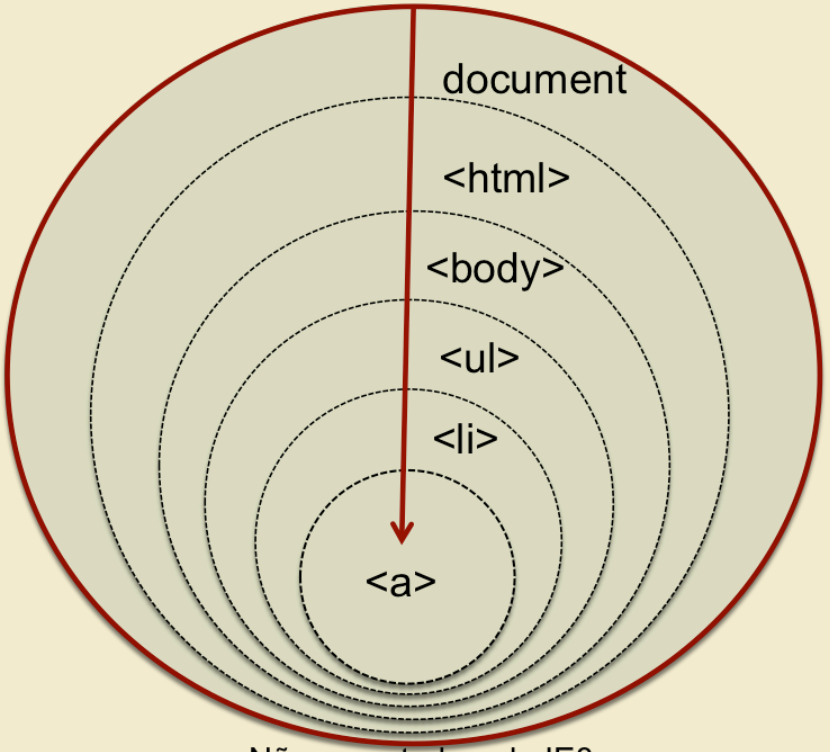
- Se um determinado evento for aplicado a diferentes elementos da árvore DOM, a captura do evento pode ser realizada de duas formas:
  - EVENT BUBBLING: a ordem de captura do evento é realizada partindo do elemento mais específico para o elemento mais geral;
  - EVENT CAPTURING: o ordem de captura do evento é realizada partindo do elemento mais geral para o elemento mais específico (não suportado pelo IE8 e versões anteriores)
- A determinação do tipo de fluxo de ativação é determinada pelo terceiro parâmetro do método addEventListener:
  `el.addEventListener(evento, funcao, tipo_ativacao)`
  - O `tipo_ativacao` assume um valor booleano:
    - `true` = event capturing
    - `false` = event bubbling (método de captura por defeito)

FLUXO DE ATIVAÇÃO DOS EVENTOS

EVENT BUBBLING



EVENT CAPTURING



Não suportado pelo IE8
e versões anteriores

O OBJECTO EVENT

- Quando um evento é disparado, leva consigo o objeto *event* que contem informação importante, nomeadamente:
  - O elemento que disparou o evento;
  - A tecla pressionada (no caso do evento *keypress*)
  - Qual a parte do browser (*viewport*) onde o utilizador clicou (no caso do evento *click*)
- O objeto é passado automaticamente pelo event handler ou listener.

O OBJECTO EVENT

- Se for necessário passar um argumento para uma função, o objecto *event* deve ser passado para a função anónima.
- Event listener sem parâmetros:
  - ```
    function verificaNomeUtilizador(e){
        var alvo = e.target;
    }
    var el = document.getElementById('utilizador');
    el.addEventListener('blur', verificaNomeUtilizador, false);
    ```
- Event listener com parâmetros:
  - ```
    function verificaNomeUtilizador(e, tam_min){
        var alvo = e.target;
    }
    var el = document.getElementById('utilizador');
    el.addEventListener('blur', function(e){
        verificaNomeUtilizador(e, 5);
    }, false);
    ```

O OBJECTO EVENT

- Existem diferenças nos métodos e propriedade do objeto *event* relativamente ao IE5-8.

Propriedade	IE5-8 equivalente	Descrição
target	srcElement	O elemento que interage com o evento
type	type	Type de evento que foi disparado
cancelable	(não suportado)	Se se pretender cancelar o comportamento por defeito do elemento
Método	**IE5-8** (propriedade equivalente)	
preventDefault()	returnValue	Termina o comportamento por defeito do evento (se puder ser cancelado)
stopPropagation()	cancelBubble	Termina o evento nos fluxos de captura bubbling e capturing

DELEGAÇÃO DE EVENTOS

- Técnica que permite em atribuir um *event listener* a um elemento contentor (p. ex.), delegando, assim, o trabalho de *event listener* ao elemento ascendente de outros elementos.
- Vantagens:
  - Poupança de memória ao não ser necessário adicionar um *event listener* a cada elemento;
  - Mais performante;
  - Apenas um função manipula n elementos;
  - Simplifica o código
  - Operações de adicionar ou remover elementos não necessitam de mais código.

MUDANÇA DO COMPORTAMENTO POR DEFEITO

- Alguns eventos, como *click* (*links*) e *submit* (*forms*), têm como comportamento por defeito saltar par outra página.
- É possível mudar o comportamento por defeito através dos métodos:
  - preventDefault() – evita o comportamento por defeito do evento
  - stopPropagation() – para interromper o event bubbling (no caso de o evento estiver propagado por elementos ascendentes)
- A aplicação destes dois métodos pode ser aplicado fazendo com que a função disparada pelo evento retorne false (*return false;*). Neste caso é evitado, ao mesmo tempo, o funcionamento por defeito e a propagação por elementos ascendentes.