ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Introduzione a Javascript II parte

Fabio Vitali

Corso di laurea in Informatica

Alma Mater – Università di Bologna

Oggi parleremo di…

Javascript

- Sintassi base (parte I)
- ***Javascript client-side (parte II)***
  - ***Oggetti predefiniti del browser***
  - ***DOM e innerHTML***
  - ***Navigazione sul DOM***
- Sintassi avanzata (parte III)

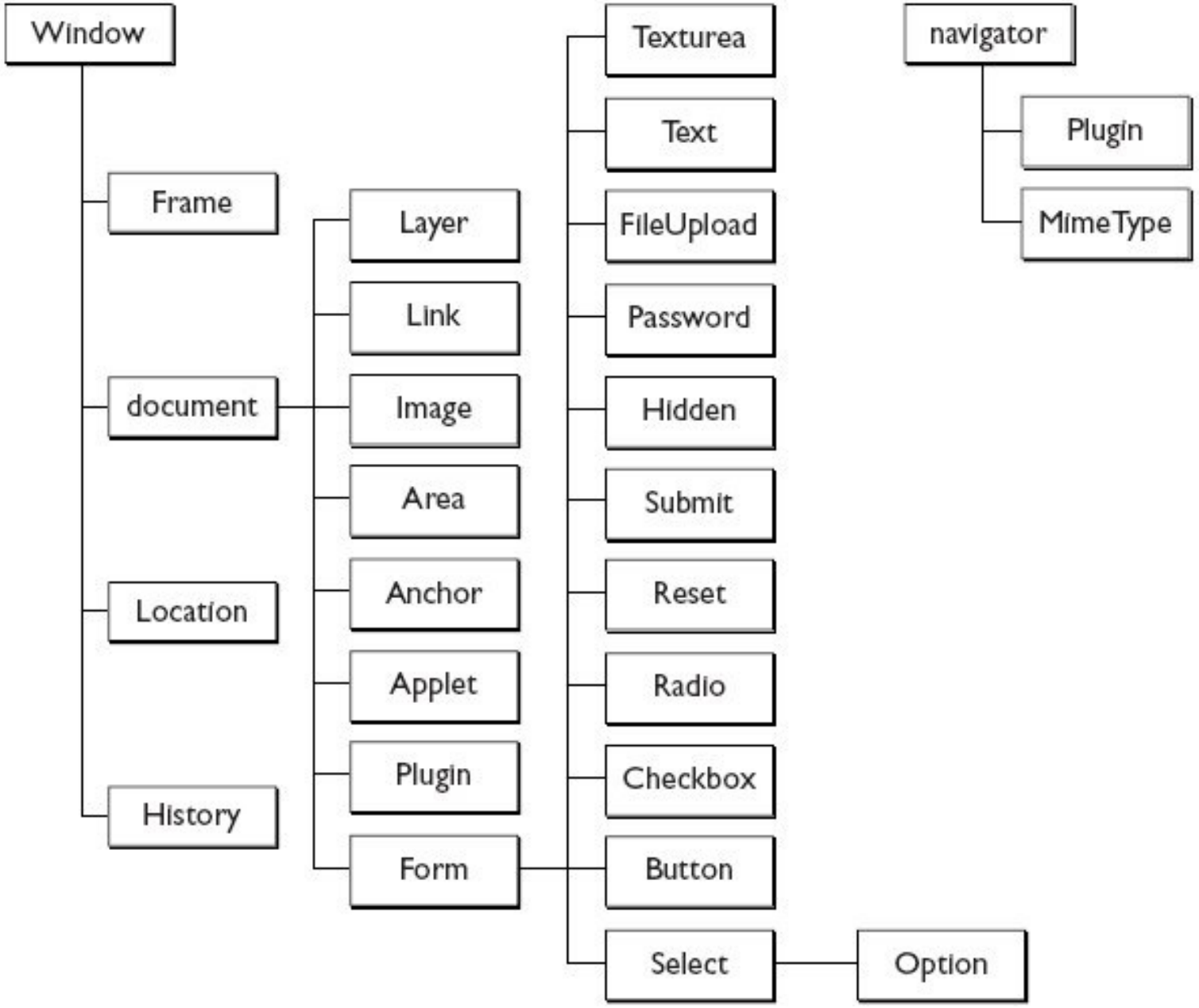
Temi trasversali

- *Navigazione sul DOM*
- *AJAX*
- *Programmazione asincrona*
- *Modularizzazione del codice*
- *Interpolazione*
- *Routing*
- *Binding mono e bi-direzionale*

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript client-side

Gli oggetti predefiniti del browser



Gli oggetti principali:
`window` e `navigator`

- **window**: è l'oggetto top-level con le proprietà e i metodi della finestra principale:
  - posizione:moveBy(x,y), moveTo(x,y), etc.
  - dimensioni: resizeBy(x,y), resizeTo(x,y), etc.
  - altre finestre: open("URLname","Windowname",["opt"])

- **navigator**: è l'oggetto con le proprietà del client come nome, numero di versione, plug-in installati, supporto per i cookie, etc.

Gli oggetti principali:
`location` e `history`

- **location**: l'URL del documento corrente. Modificando questa proprietà il client accede a un nuovo URL (redirect):
  - `window.location ="http://www.cs.unibo.it/";`
  - `window.location.href ="http://www.cs.unibo.it/";`

- **history**: l'array degli URL acceduti durante la navigazione. Possibile creare applicazioni client-side dinamiche che 'navigano la cronologia':
  - Proprietà: length, current, next
  - Metodi: back(), forward(), go(int)

Gli oggetti principali:
document

document rappresenta il contenuto del documento, ed ha proprietà e metodi per accedere ad ogni elemento nella gerarchia:

- document.title: titolo del documento
- document.forms[0]: il primo form
- document.forms[0].checkbox[0]: la prima checkbox del primo form
- document.forms[0].check1: l'oggetto con nome "check1" nel primo form (non per forza una checkbox!)
- document.myform: l'oggetto "myform"
- document.images[0]: la prima immagine

- Inoltre **document** rappresenta l'oggetto DOMDocument del DOM del documento visualizzato

Modello di documento

Ogni oggetto nella gerarchia è caratterizzato da un insieme di proprietà, metodi ed eventi che permettono di accedervi, controllarlo, modificarlo.

Javascript nacque per controllare i valori di un form:

```
function verify() {
        if (document.forms[0].elements[0].value == ""){
                alert("Il nome è obbligatorio!")
                document.forms[0].elements[0].focus();
                return false;
        }
        return true;
}


<form action= "…"  onSubmit="Verify()">
<p>Name: <input type="text" name="userName"> </p>
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Il Document Object Model (DOM)

Document Object Model

Adesso parliamo di:

- Struttura programmatica del documento HTML
- I principali oggetti del DOM
- Manipolazione del DOM

Una questione spinosa: il parsing di HTML 5

- Il WhatWG ha definito una volta per tutte i meccanismi di parsing ed interpretazione del codice HTML.
- Le specifiche “HTML Living Standard” definiscono un algoritmo (piuttosto complesso) per fare il parsing di qualunque documento HTML, anche dei documenti mal formati, sulla base di ciò che i browser già facevano.
- In realtà dalla prospettiva WAI questi documenti non sono propriamente “mal formati” ma semplicemente “non strict”. Sono validi a tutti gli effetti, tanto quanto i documenti XHTML!
- Pragmaticamente potremmo dire: “l’importante è arrivare ad una struttura dati in memoria unica su cui costruire applicazioni”. E' a questo scopo che la vera attenzione da parte del WHATWG è la costruzione di una struttura dati chiamata Document Object Model (DOM), a cui (in maniera più o meno precisa) sia possibile arrivare a partire dalla stringa HTML e da cui si possa generare nuovamente una altra stringa HTML.

Un problema risolto o uno in più?

- Aver uniformato l'algoritmo di parsing non è un deterrente per creare pagine "ben formate", al contrario lascia maggiore libertà e margine di errore agli sviluppatori.
- Se esistono pagine errate è perché, in primo luogo, ci sono stati sviluppatori che hanno creato pagine non valide e usato il "funziona sul mio browser" come strumento di convalida.
- Questa brutta pratica è nata dal fatto che, in mancanza di regole semplici (SGML) i browser hanno sempre accettato tutti i documenti e fatto del loro meglio per visualizzarli.
- Il vero problema è che in questo modo prolifereranno le pagine non corrette e sarà più complesso estrarre i dati e implementare manipolazioni automatiche dei contenuti.
- La cosa si complicherà ancora di più quando (coi sistemi a componenti) lo stesso concetto di markup perderà valore a vantaggio di sintassi miste Javascript/Markup/CSS/whatever create ad hoc e mai standardizzate.

Il Document Object Model

Il Document Object Model è un **interfaccia di programmazione (API)** per documenti sia HTML sia XML.

Definisce la struttura logica dei documenti ed il modo in cui si accede e si manipola un documento.

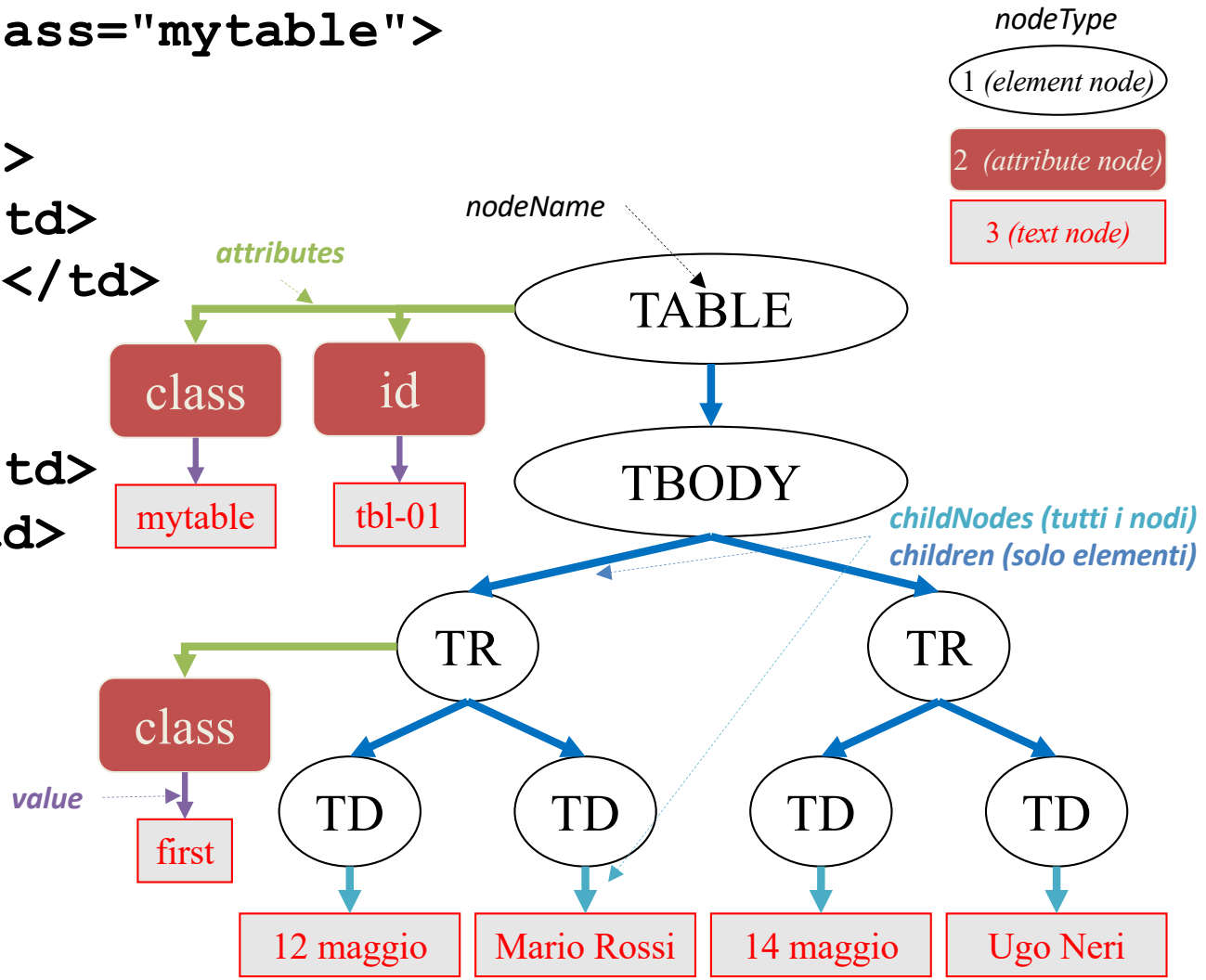
Utilizzando DOM i programmatori possono costruire documenti, navigare attraverso la loro struttura, e aggiungere, modificare o cancellare elementi.

Ogni componente di un documento HTML o XML può essere *letto, modificato, cancellato o aggiunto* utilizzando il Document Object Model.

Struttura di un DOM

Sia dato un documento HTML come il seguente:

```
<table id="tbl-01" class="mytable">
  <tbody>
    <tr class="first">
      <td>12 maggio</td>
      <td>Mario Rossi</td>
    </tr>
    <tr>
      <td>14 maggio</td>
      <td>Ugo Neri</td>
    </tr>
  </tbody>
</table>
```



Oggetti del DOM

Il core del DOM definisce alcune classi fondamentali per i documenti HTML e XML, e ne specifica proprietà e metodi.

La classe principale di DOM è *DOMNode*, di cui la maggior parte delle altre classi è una sottoclasse.

Le classi principali definiti nel DOM sono:

- DOMDocument : il documento di cui si sta parlando
- DOMElement: ogni singolo elemento del documento
- DOMAttr: ogni singolo attributo del documento
- DOMText: ogni singolo nodo di testo del documento
- DOMComment, DOMProcessingInstruction, DOMCDATASection, DOMDocumentType, ecc.

DOM Node

DOMNode specifica i metodi per accedere a tutti gli elementi di un nodo di un documento, inclusi il nodo radice, il nodo documento, i nodi elemento, i nodi attributo, i nodi testo, ecc. Semplificando:

membri		**metodi**
– nodeName	*(uppercase string)*	insertBefore()
– nodeType	*(number)*	replaceChild()
– children	*(array)*	removeChild()
– childNodes	*(array)*	appendChild()
– parentNode	*(elementNode)*	hasChildNodes()
– attributes	*(array)*	hasAttributes()

N.B.: non è vero! Sono HTMLCollection, NodeList e NamedNodeMap rispettivamente, ma Array.from() restituisce un array.

DOM Document

DOMDocument specifica i metodi per accedere al documento principale, tutto compreso. È equivalente alla radice dell'albero (non all'elemento radice!!!). Semplificando:

membri	**metodi**
docType	createElement()
documentElement	createAttribute()
	createTextNode()
	getElementsByTagName()
	getElementById()

DOM Element

DOMElement specifica i metodi e i membri per accedere a qualunque elemento del documento. Semplificando:

membri	**metodi**
nodeName	getAttribute()
	setAttribute()
	removeAttribute()

… e analogamente per le altre classi ed interfaccia del DOM.

Javascript e DOM

Javascript implementa i metodi standard per accedere al DOM del documento.

```
var c = document.getElementById(‘c35');

c.setAttribute(‘class',‘prova1');
c.removeAttribute(‘align') ;

var newP = document.createElement('p');

var text = document.createTextNode('Ciao Mamma.');

newP.appendChild(text);

c.appendChild(newP);
```

```
// Creazione elementi  singoli
Olist = document.createElement("ol");
voce1 = document.createElement("li");
voce2 = document.createElement("li");
testo1 = document.createTextNode(“un po’ di testo");
testo2 = document.createTextNode(“altro testo – item 2");

// Creazione lista completa

voce1.appendChild(testo1);
voce2.appendChild(testo2);
Olist.appendChild(voce1);
Olist.appendChild(voce2);

// Inserimenti lista in una data posizione

div = document.getElementById("lista");
body = document.getElementsByTagName("body").item(0);
body.insertBefore(Olist,div);
```

innerHTML e outerHTML

Il DOM per HTML (non generale!) permette di leggere/scrivere interi elementi, trattandoli come stringhe:

- **innerHTML**: legge/scrive il contenuto di un sottoalbero (escluso il tag dell'elemento radice)
- **outerHTML**: legge/scrive il contenuto di un elemento (incluso il tag dell'elemento radice)

```
// sia dato un HTML:
<div id="p1">
  <p>Paragrafo!</p>
</div>

Accesso ai valori attuali:
let a = document.getElementById("p1");
let b = a.innerHTML;                    // -> <p>Paragrafo!</p>
let c= a.outerHTML;                     // -> <div
id="d"><p>Paragrafo!</p></div>
Modifica dei valori attuali:
a.innerHTML = "<ul><li>Lista!</li></ul>";
```

Selettori in DOM

I metodi standard in DOM per accedere ai nodi di un documento sono essenzialmente:

- **getElementById**: solo ovviamente se l'elemento ha un id
- **getElementsByName**: se l'elemento ha un attributo name
- **getElementsByTagName**: tutti gli elementi con nome specificato

Il successo di JQuery ha portato nel tempo a proporre ed implementare in DOM HTML anche tre nuovi selettori:

- **getElementsByClassName**; cerca tutti gli elementi di classe specificata
- **querySelector**: accetta un qualunque selettore CSS e restituisce il primo elemento trovato - del tutto equivalente a $()[0] in JQuery
- **querySelectorAll**: accetta un qualunque selettore CSS e restituisce tutti gli elementi trovati - del tutto equivalente a $() in JQuery

Javascript ed eventi DOM

Javascript permette di associare callback di eventi ad oggetti (dichiarazione locale o globale)

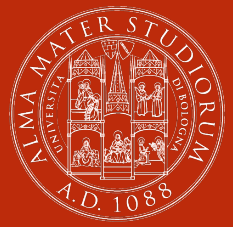
```
...
<script language="JavaScript">
  window.onkeypress= pressed;
  window.document.onclick = clicked;

  function pressed(e) { alert("Key pressed: " + e.which);}
  function clicked() { alert("Mouse Click! "); }
</script>
...
<body>
  <p id="mypara">Puoi
    <a href="test.htm" onClick="alert('Link!');">
      cliccare qui
    </a>
    oppure qui.
  </p>
</body>
```

Un piccolo esercizio

Andiamo a creare una calcolatrice digitale

Troviamo tutto a:

`http://www.fabiovitali.it/TW/2024/calc/`



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fabio Vitali

Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna

Fabio.vitali@unibo.it

www.unibo.it