JavaScript

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Recap

- Domande su HTTP?
- Domande su HTML?
- Domande su tecnologie XML?

Mappa Concettuale della Lezione

Tipi di Documenti Web

- Statici.
  - File associati al web server il cui contenuto non cambia. Tutte le richieste di accesso conducano alla visualizzazione della stessa informazione.
- Dinamici.
  - Non esistono in formato predefinito, ma sono generati dal web server ogni volta che un browser ne fa richiesta.
  - La generazione consiste nella esecuzione di un programma da parte del server. Il risultato cioè il documento è trasmesso dal server al browser.
  - I contenuti visualizzati possono variare da una richiesta all'altra.
- Attivi
  - Sono applicazioni che il server invia al browser richiedente che vengono eseguite dal browser. Il risultato rappresenta il documento attivo.
  - I contenuti visualizzati possono cambiare anche in funzione dell'interazioni con l'utente

Confronto fra i Tipi di Documento

- Statici
  - Vantaggi: semplicità, affidabilità e rapidità di acquisizione e visualizzazione. Possibilità per i browser di depositarne una copia nella cache.
  - Svantaggi: poca flessibilità, il loro aggiornamento deve essere fatto manualmente.

- Dinamici
  - Vantaggi: capacità di riportare informazioni aggiornate. Adatti a registrare informazioni fluttuanti (quotazioni di borsa, previsioni meteorologiche,disponibilità di biglietti per un concerto).
  - Responsabilità in carico ai server. Nessuna distinzione per i browser tra documenti statici e dinamici che ricevono un documento HTML.
  - Svantaggi: maggiori costi e incapacità dei documenti di mostrare l'informazione in divenire una volta che questa è visualizzata (quotazioni di borsa).

Confronto fra i Tipi di Documento

- Attivi
  - Vantaggi: Capacità di aggiornarsi continuamente: può accedere direttamente alle informazioni e aggiornare continuamente i dati
  - visualizzati (documento attivo dedicato alle quotazioni di borsa).
  - Svantaggi:scrittura del codice in modo indipendente alla piattaforma. Sicurezza

Realizzazione di Documenti Dinamici

Modifiche al server web:

- Deve eseguire un'applicazione che generi il documento in presenza di una richiesta. Il server riceve l'output dell'applicazione e lo invia al browser.
- È necessario predisporre una nuova applicazione per ciascun documento dinamico.
- Il server deve essere in grado di distinguere le stringhe URL che denotano documenti statici da quelle dei documenti dinamici e in quest'ultimo caso deve determinare l'applicazione da eseguire per la generazione del documento.

Realizzazione di Documenti Attivi: Problemi

- Documenti attivi
- Documenti dinamici: impossibilità di modificare le informazioni contenute dopo la loro visualizzazione (es.,non possono contenere animazioni).

Soluzioni al problema

- Server push. Il server deve eseguire il programma per la generazione del documento a ciclo continuo inviando i risultati al browser.
- Dal punto di vista dell'utente i contenuti della pagina continuano a cambiare mentre la si esamina.
- In presenza di molti client che tentano di accedere allo stesso documento, il server rischia il sovraccarico.

Documenti Attivi: Soluzioni

- Documenti attivi. Sono i browser ad eseguire l'elaborazione necessaria ad aggiornare il documento.
- Quando i server ricevono la richiesta di un documento attivo inviano al browser un programma che viene eseguito localmente.
- Browser e server trattano i documenti attivi al pari di quelli statici per ciò che riguarda il loro trasferimento.
- Il browser può conservarne una copia nella cache.
- I documenti attivi non contengono tutto il software necessario alla loro esecuzione, perché parte di quest'ultimo è detenuto staticamente dal browser (interpreti, macchine virtuali,..).
- I documenti attivi sono originariamente scritti in codice sorgente .Il compilatore produce una forma eseguibile che viene inviata su richiesta al browser.
- L'esecuzione nel browser richiede la presenza di opportuno software per collegare le librerie richieste.

Cosa è JavaScript?

- JavaScript è un linguaggio di scripting sviluppato per dare interattività alle pagine HTML
- JavaScript è un linguaggio interpretato (interpretazione del sorgente testuale)
- JavaScript è solitamente inserito direttamente nelle pagine Web

Java e JavaScript

- Java è JavaScript sono profondamente differenti.
- JavaScript ha obiettivi e campo di applicazione normalmente diverso e più limitato rispetto a Java. In particolare JavaScript è IL linguaggio per la programmazione client side di applicazioni web
- JavaScript è:
  - linguaggio interpretato
  - Object-based
  - Ha una gestione dinamica dei tipi
  - Nasce per rendere maggiormente interattive le pagine Web

JavaScript e le sue Standardizzazioni

- Il linguaggio JavaScript nasce dallo sforzo di Netscape
- In seguito anche Microsoft ha lavorato sul linguaggio producendo una versione del linguaggio chiamata JScript
- JavaScript è divenuto standard ECMA (ECMA-262) ed ha preso il nome di ECMAscript
- JavaScript è divenuto anche uno standard ISO (ISO/IEC 16262)

Cosa possiamo fare con JavaScript?

- JavaScript è IL linguaggio che viene usato nella programmazione client-side
- Nasce per dare dinamicità alle pagine Web
  - Possiamo accedere e modificare elementi della pagina HTML
  - Reagire ad eventi generati dall'interazione fra utente e pagina (es. visita di un link)
  - Possiamo validare i dati inseriti dall'utente
  - Possiamo interagire con il browser, es. determinare il broser dell'utente o lavorare con i cookie

Come inseriamo JavaScript nelle pagine HTML?

- Il codice JavaScript è inserito direttamente nelle pagine HTML

Il tag <script> ci consente di inserire del codice per script client-side

L'attributo type specifica che lo script è JavaScript

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello World!");
    </script>
  </body>
</html>
```

Document.write("…") ci consente di scrivere l'output nella pagina

CONTROLLARE Problemi con JavaScript

- Non tutti i browser potrebbero supportare JavaScript (anche se questo è sempre meno vero)
- Non tutti gli utenti potrebbero volere permettere che i loro browser eseguano gli script JavaScript

TRUCCO

Se non eseguo javascript il browser esegue il commento. Se invece uso javaScript // evita la valutazione di -->

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Hello World!");
//-->
</script>
</body>
</html>
```


Altro su inclusione di JavaScript nell'HTML

Trucco alternativo ed equivalente al precedente per non mostrare il codice nel caso di script disabilitati…

```
<html>
<body>
<script type="text/javascript">
//<![CDATA[
      document.write("Hello World!");
//]]>
</script>
      <noscript>Attiva JavaScript</noscript>
</body>
</html>
```

Contenuto alternativo nel caso in cui gli script non siano abilitati. Può essere qualunque cosa lecita in un body HTML

Dove Inserire il Codice JavaScript

1) Script nella sezione head del documento HTML. Consente di eseguire lo script solo quando questi viene chiamato, o a seguito del verificarsi di un evento. Garantisce che lo script sia stato caricato completamente prima di iniziarne l'esecuzione

```
<head>
   …
    <script type="text/javascript">....</script>
    …
</head>
<body> … </body>
```


Dove Inserire il Codice JavaScript

2) Script nella sezione body del documento HTML. Lo script viene eseguito al caricamento della pagina e tipicamente consente di generarne (parte) del contenuto

```
<head>… </head>
<body>
    …
    <script type="text/javascript">....</script>
    …
</body>
```

Dove Inserire il Codice JavaScript

3) Sia nella head che nel body del documento HTML. Non sono posti vincoli su numero di script JavaScript che può contenere una pagina. La scelta sulla locazione in cui inserire il codice dipende dall'applicazione

```
<head>
    …
    <script type="text/javascript">....</script>
    …
</head>
<body>
    …
    <script type="text/javascript">....</script>
    …
</body>
```


Dove Inserire il Codice JavaScript

4) Possiamo inserire anche il codice JavaScript in un file esterno (tipicamente un file .js). Il codice può essere inserito nella head e/o nel body del documento HTML. Valgono le stesse regole viste prima. Molto comodo se diverse pagine usano lo stesso script

Specifica la URL a cui reperire lo script

```
<head>… </head>
<body>
    …
    <script type="text/javascript" src="jquery.js">
    </script>
    …
</body>
```

Statement JavaScript

- Gli statement JavaScript sono comandi indirizzati al browser. Gli statement vengono eseguiti sequenzialmente

```
<script type="text/javascript">
   document.write("<h1>Titolo</h1>");
   document.write("<p>Paragrafo</p>");
</script>
```

Lo script chiede al browser di inserire un titolo "Titolo" con heading H1 e un paragrafo "Paragrafo" nella pagina

Blocchi di Istruzioni

Analogamente ad altri linguaggi quali C/C++ o Java, JavaScript permette di definire blocchi di istruzioni da eseguire in modo sequenziale

```
<script type="text/javascript">
        {
        document.write("<h1>Titolo</h1>");
        document.write("<p>Paragrafo</p>");;
        }
</script>
```

Commenti

La sintassi dei commenti è la stessa di molti altri linguaggi di programmazione

```
<script type="text/javascript">
// Commento
document.write("<h1>Titolo</h1>");
// altro commento
document.write("<p>paragrafo</p>"); //altro commento
document.write("<p>altro paragraph</p>");
/* commento
document.write("<p>paragrafo</p>");
document.write("<p>paragrafo</p>");
document.write("<p>paragrafo</p>");
*/
</script>
```


Variabili

- Le variabili in JavaScript possono essere usati per memorizzare valori o espressioni
- Le variabili JavaScript sono case-sensitive e devono iniziare con una lettera o un underscore “ ”
  –

```
var pippo;
var pluto=5;
var topolino=“ciao”;
var minni=7

pippo=pluto+minni
```

Variabile vuota

La keyword var consente di dichiarare una variabile

Javascript: Valori e Variabili

- Javascript riconosce i seguenti tipi di valori:
  - Numerici (interi o floating point)
  - Booleani (true or false)
  - Stringhe
  - null
  - undefined
- Javascript è un linguaggio a tipizzazione dinamica:

```
var answer = 42;
answer = "Thanks for all the fish...“;
```

- Variabili:
  - Il nome deve iniziare con una lettera o con _
  - Javascript è case sensitive
  - Assegnazione: x=42; var x=42;
  - Una variabile non assegnata è undefined
  - Scope: global,local

Operatori Aritmetici

Operatore	Descrizione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Modulo
++	Incremento
--	Decremento

Operatori di Assegnamento

Operatore	Descrizione
=	Assegnamento
+=	x+=1 → x=x+1
-=	x-=1 → x=x-1
*=	x*=2 → x=x*2
/=	x/=2 → x=x/2
%=	x%=2 → x=x%2

Operatori Logici

&&	`expr1 && expr2`	(Logical AND) Returns `expr1` if it can be converted to false; otherwise, returns `expr2`. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	`expr1 || expr2`	(Logical OR) Returns `expr1` if it can be converted to true; otherwise, returns `expr2`. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	`!expr`	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Operatori di Confronto

Operator	Description	Examples returning true

Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	`3 == var1`
		`"3" == var1`
		`3 == '3'`

Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	`var1 != 4` `var2 != "3"`
Strict equal (===)	Returns true if the operands are equal and of the same type.	`3 === var1`
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	`var1 !== "3"` `3 !== '3'`
Greater than (>)	Returns true if the left operand is greater than the right operand.	`var2 > var1`
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (<)	Returns true if the left operand is less than the right operand.	`var1 < var2`
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

Operatori bit a bit

Operator	Usage	Description
Bitwise AND	*a* & *b*	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	*a* | *b*	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	*a* ^ *b*	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	~ *a*	Inverts the bits of its operand.
Left shift	*a* << *b*	Shifts a in binary representation b bits to left, shifting in zeros from the right.
Sign-propagating right shift	*a* >> *b*	Shifts a in binary representation b bits to right, discarding bits shifted off.
Zero-fill right shift	*a* >>> *b*	Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

if…else…

- In JavaScript possiamo utilizzare strutture di controllo analoghe a quelle che conosciamo in altri linguaggi
  - if (condizione) {…}
  - if (condizione) {…} else {…}
  - if (condizione1) {…} else if (condizione2) {…} else {…}

switch

- Analogamente ad altri linguaggi di programmazione è disponibile lo statement switch

Il Ciclo for

Anche il ciclo for ha una sintassi comune ad altri linguaggi

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0; i<=10; i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```


Il Ciclo while

Analogo discorso per il ciclo while

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{document.write("The number is " + i);
document.write("<br />");i=i+1;}
</script>
</body>
</html>
```

Il Ciclo do…while

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{document.write("The number is " +i);
document.write("<br />");
i=i+1;}
while (i<0);
</script>
</body>
</html>
```


Il Ciclo for… in

- Usato per iterare gli elementi di un array sulla base delle proprietà degli oggetti
- Il codice nel corpo del ciclo for… in è eseguito una volta per ogni elemento/proprietà

for (variable in object)

{*codice da eseguire*}

Esempio di Ciclo for… in

```
<html>
<body>
<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = ”Panda"; mycars[1] = ”Uno";
mycars[2] = ”Punto"; mycars[2] = ”Clio";
for (x in mycars)
{document.write(mycars[x] + "<br />");}
</script>
</body>
</html>
```

Definizione di un Array e dei suoi elementi

Popup Box

Alert Box

alert(”Occhio alla Penna!");

Si usa quando vogliamo essere certi di notificare delle informazioni all’utente.

Confirm Box

confirm(”Sicuro sicuro??");

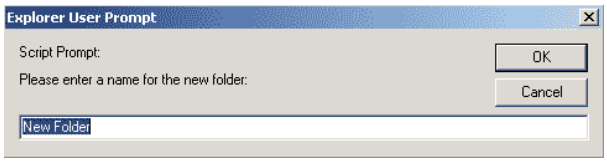
Per verificare se l’utente accetta una condizione. Restituisce true se l’utente preme OK, false altrimenti

Popup Box

Prompt Box

prompt(”testo”, “valore di default”);

Se l’utente conferma con OK il popup restituisce il valore che l’utente ha specificato, null viceversa



Funzioni in JavaScript

- Per evitare che il browser esegua uno script durante il caricamento delle pagine è possibile mettere lo script in una funzione
- Un funzione contiene codice che verrà eseguito a seguito di un evento o di una esplicita chiamata alla funzione
- Se vogliamo essere sicuri che la funzione sia stata completamente caricata dalla pagina è bene mettere la definizione nella head

Sintassi per la Definizione di Funzioni

```
function functionname (var1, var2, ..., varN)
{
        codice della funzione
}

function functionname (var1, var2, ..., varN)
{
        codice della funzione
        return val;
}
```

Valgono le solite regole per lo scope delle variabili locali dichiarate nella funzione

Javascript: Funzioni predefinite

- eval(expr)
  valuta la stringa expr (espressione,istruzioni)
- isFinite(number)
  number=NaN,+inf,-inf → false
- isNaN(testValue)
  testValue = NaN → true
- parseInt(str, [radix])
  converte la string str in un intero in base radix
- parseFloat(str)
  converte la string str in un floating point

Gestione degli Eventi Generati dal Browser

- JavaScript è un linguaggio semplice
- JavaScript non supporta multi-threading ma fornisce un meccanismo che consente al browser di invocare una funzione qualora si verifichino specificati eventi
  - Click sul mouse
  - Il caricamento di una pagina web o di una immagine,
  - Il movimento del puntatore del mouse su aree specificate
  - la submission di una form
  - …
- Gli eventi sono associati a funzioni (handler) che vengono invocate solo a seguito del verificarsi dell'evento

Javascript: Eventi

Event	Applies to	Occurs when	Event handler
Load	document body	User loads the page in the Navigator	onLoad
MouseDown	documents, buttons, links	User depresses a mouse button	onMouseDown
MouseMove	nothing by default	User moves the cursor	onMouseMove
MouseOut	areas, links	User moves cursor out of a client-side image map or link	onMouseOut
MouseOver	links	User moves cursor over a link	onMouseOver
MouseUp	documents, buttons, links	User releases a mouse button	onMouseUp
Move	windows	User or script moves a window	onMove
Reset	forms	User resets a form (clicks a Reset button)	onReset
Resize	windows	User or script resizes a window	onResize
Select	text fields, textareas	User selects form element's input field	onSelect
Submit	forms	User submits a form	onSubmit
Unload	document body	User exits the page	onUnload

onload e onUnload

- Gli eventi onload ed onUnload sono generati quando l'utente entra o lascia una pagina
- L'evento onload viene normalmente usato per controllare tipo e versione del browser al fine di adattare l'erogazione dei contenuti
- Spesso questi eventi sono anche utilizzati per accedere ai cookie e adattare di conseguenza la pagina sulla base di informazioni relative all'utente, es. pagina di wellcome con il nome

onFocus, onBlur ed onChange

- Gli eventi onFocus, onBlur ed onChange sono normalmente impiegati nella validazione delle form.
- Esempio

```
<input type="text" size=”20” id=”pippo"
                          onchange=”funzione()">
```

Quando l'utente cambia il contenuto della form viene invocata la funzione

onSubmit

- È molto usato e consente di validare tutti i campi di una form prima che i dati inseriti dall'utente vengano inoltrati al server

```
<form method="post" action="xxx.htm"
   onsubmit="return checkForm()">
```

Quando l'utente ha compilato la form e conferma l'operazione viene invocata la funzione associata. Se la funzione restituisce true allora i dati saranno inviati al server. Se viceversa la funzione restituisce false i dati non saranno inviati al server

OnMouseOver ed OnMouseOut

- Spesso sono usati per creare bottoni animati…

```
<a href="mypage.htm"
 onmouseOver="MouseOverRoutine('button1')"
 onmouseOut="MouseOutRoutine('button1')">
 <Img Src="button1a.gif" name="button1" ></a>
```

Immagini Animate…

```
<head>
<script> <!--
 button1up = new Image;
 button1up.src = "button1a.gif";
 button1down = new Image;
 button1down.src = "button1b.gif";
 function MouseOverRoutine(ButtonName)
    { if (ButtonName=="button1")  {document.button1.src = button1down.src;} }
 function MouseOutRoutine(ButtonName)
    { if (ButtonName=="button1")  {document.button1.src = button1up.src;} }
-->
</script>
</head> <body> …
 <a href="mypage.htm" onmouseOver="MouseOverRoutine('button1')"
    onmouseOut="MouseOutRoutine('button1')"> <Img Src="button1a.gif"
    name="button1" ></a> ….
 </body>
```


Gestione delle Eccezioni

- La gestione delle eccezioni in JavaScript ha una sintassi analoga a quella di Java

```
try {
Codice che si prova ad eseguire
} catch(err){
    Codice per la gestione della eccezione
  }
```

Eccezioni ed Errori

- L'esempio riporta una sezione di codice che dovrebbe consentire la stampa della stringa "ciao"

```
<html>
<head>
    <script type="text/javascript">
    var txt=""function message(){
            try  {  adddlert("Welcome guest!");
            }catch(err)  {  txt="There was an error on this page";
                                txt+="Error description: " +
                                err.description + "\n";
                                txt+="Click OK to continue.";
                                alert(txt);  }}
    </script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body></html>
```

La funzione adddlert non viene riconosciuta da JavaScript. Viene generata una eccezione gestita dal codice nel blocco catch

Sollevare una Eccezione

- Analogamente a Java è possibile sollevare eccezioni. La sintassi è la seguente

throw (eccezione);

Eccezione potrebbe essere una stringa, un intero, un boolean oppure un oggetto.

Esempio di Eccezione

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try{ if(x>10) throw "Err1";
      else if(x<0)throw "Err2";}
catch(er) {
   if(er=="Err1") alert("Error! The value is too high");
   if(er=="Err2") alert("Error! The value is too low");
}
</script>
</body>
</html>
```

Uso tipico delle eccezioni

L'Evento onError

- Nel caso in cui si verifichi un errore il browser genera l'evento onError a cui è possibile associare un handler

```
onerror=handleErrfunction
function handleErr(msg, url, l){
   funzione per la gestione dell'errore
   return (true oppure false);
}
```

Vengono passati alla funzione rispettivamente il messaggio di errore, la URL della pagina che lo ha causato e la linea a cui l'errore si è manifestato

Se true il browser non visualizza l'errore. Se viceversa è false il browser visualizza l'errore nella console JavaScript

Esempio di uso di onError

```
<html><head><script type="text/javascript">

onerror=handleErr;
var txt="";
function handleErr(msg,url,l)
{
txt="There was an error on this page.\n\n";
txt+="Error: " + msg + "\n";
txt+="URL: " + url + "\n";
txt+="Line: " + l + "\n\n";
txt+="Click OK to continue.\n\n";
alert(txt);
return true;
}

function message() {adddlert("Welcome guest!");   ///// Errore qui}
</script>

</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```


Oggetti in JavaScript

- La gestione degli oggetti in JavaScript è semplificata rispetto a linguaggi come Java, C# o C++
- La sintassi per accedere a attributi ed invocare metodi è analoga a quelle che dei principali linguaggi

```
<script type="text/javascript">
var str="Ciaooo";
document.write(str.length);
document.write(str.toUpperCase());
</script>
```

Attributo della stringa str

Invoca il metodo toUpperCase() della stringa str

Definizione di Classi in JavaScript

- Ci sono diversi modi per definire oggetti in JavaScript
  - Definizione diretta
  - Definizione di un template

Definizione Diretta

La definizione diretta permette di definire ed istanziare un oggetto. È ragionevole applicarla solo per semplici script. Es.

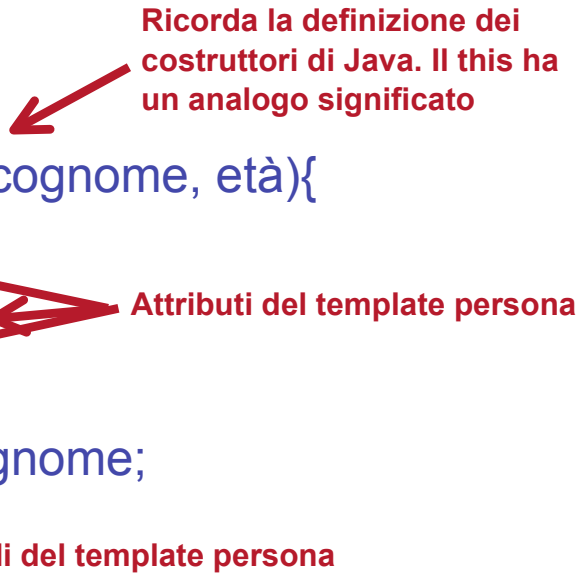
```
Person=new Object();
Person.nome="Car Carlo";
Person.cognome="Pravettoni";
Person.età=50;
Person.metodo=metodo;
```

Creo una istanza di Object e aggiungo gli attributi nome, cognome ed età

Associo il metodo metodo() all'oggetto che ho definito

Definizione di un Template

Il template è sempre una funzione in JavaScript



Definizione di un Template

```
function setCognome (nuovoCognome){
this.cognome=nuovoCognome;
}
```

Per istanziare un oggetto si usa una sintassi simile a quella di Java

```
pers=new persona("Homer", "Simpson", 99);
```

Javascript: Operatori Speciali

- new: crea un'istanza di un oggetto
- this: referenzia l'oggetto corrente
- typeof: restituisce una stringa che rappresenta il tipo di un oggetto
- void: valuta una espressione javascript senza restituire alcun valore. Es. usata per i link a pagine web

```
<A HREF="javascript:void(0)">Click here to do nothing</A>

<A HREF="javascript:void(document.form.submit())">
Click here to submit</A>
```


Oggetti di uso Comune

- String
- Array
- Date
- Boolean
- Math
- RegExp

Vedere la documentazione JavaScript per maggiori informazioni

Javascript: Oggetto Array

Creazione di un Array

```
arrayObjectName = new Array(element0, element1, ..., elementN)
arrayObjectName = new Array(arrayLength)
```

Popolamento di un Array

```
emp[1] = "Casey Jones"
emp[2] = "Phil Lesh"
emp[3] = "August West“
myArray = new Array("Hello", myVar, 3.14159)
```

Riferimento ad un elemento

```
myArray[0]
```

E’ possibile definire Array multidimensionali

Esempi: Determinare il Browser

- Spesso è necessario adattare gli script alle caratteristiche del browser utilizzato dall’utente. La ragione deriva da differenze nella implementazione dei diversi browser.
- Navigator è un oggetto cui possiamo accedere che contiene diverse informazioni di interesse quali:
  - il nome del browser (appName)
  - la versione (appVersion)
  - …

Esempi: Determinare il Browser

```
<html>
<head>
<script type="text/javascript">
function detectBrowser() {
var browser=navigator.appName;
var version=navigator.appVersion;   // contiene varie informazioni la funzione
var vers=parseFloat(version)        // parseFloat restituisce la sola versione…

If ((browser=="Netscape"||browser=="Microsoft Internet Explorer") && (vers>=5))
{alert("browser OK");}
else {alert("browser non supportato");}
}

</script>
</head>
<body onload="detectBrowser()">
</body>
</html>
```

Al caricamento della pagina il browser invoca la funzione detect browser che utilizza navigator per stampare un avviso per mezzo di un popup

Esempio: Validare una input form

- Riduce il carico delle applicazioni server side filtrando l'input
- Riduce il ritardo in caso di errori di inserimento dell'utente
- Semplifica le applicazioni server side
- E' possibile validare un documento in due momenti:
  - Durante l'inserimento utilizzando l'evento onChange
  - Al momento del submit della form

Esempio: Validare una input form

```
head>
<script>
  function isaPosNum(s) {
    return (parseInt(s) > 0)
  }

  function qty_check(item, min, max) {
    var returnVal = false;
    if (!isaPosNum(item.value))
      alert("Please enter a positive number");
    else if (parseInt(item.value) < min)
      alert("Please enter a " + item.name + " greater than " + min);
    else if (parseInt(item.value) > max)
      alert("Please enter a " + item.name + " less than " + max);
    else
      returnVal = true;
    return returnVal;
  }

  function validateAndSubmit(theform) {
    if (qty_check(theform.quantity, 0, 999)) {
      alert("Order has been Submitted");
      return true;
    } else {
       alert("Sorry, Order Cannot Be Submitted!");
       return false;
    }
  }
</script>
/head>
```


Esempio: Validare una input form

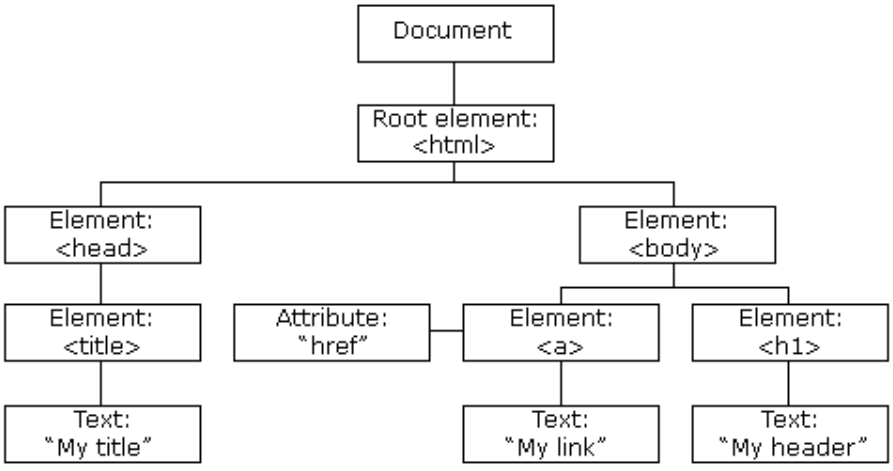
```
<body>
  <form name="widget_order" action="lwapp.html" method="post">
    How many widgets today?
    <input type="text" name="quantity" onchange="qty_check(this, 0, 999)">
    <br>
    <input type="submit" value="Enter Order“
              onclick="validateAndSubmit(this.form)">
  </form>
</body>
```

```
<form name="widget_order" action="lwapp.html" method="post“
          onSubmit="return qty_check(theform.quantity, 0, 999)">
  ...
  <input type="submit">
  ...
</form>
```

DOM

Il Modello DOM e le Pagine HTML

- Il modello DOM rappresenta una pagina HTML come un albero



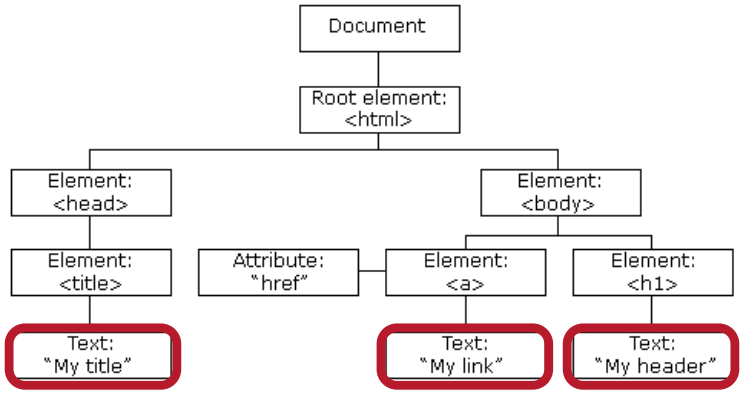
Il Modello DOM e le Pagine HTML

Il modello DOM è stato visto nella precedente lezione ed i suoi concetti di base si applicano anche alle pagine (X)HTML:

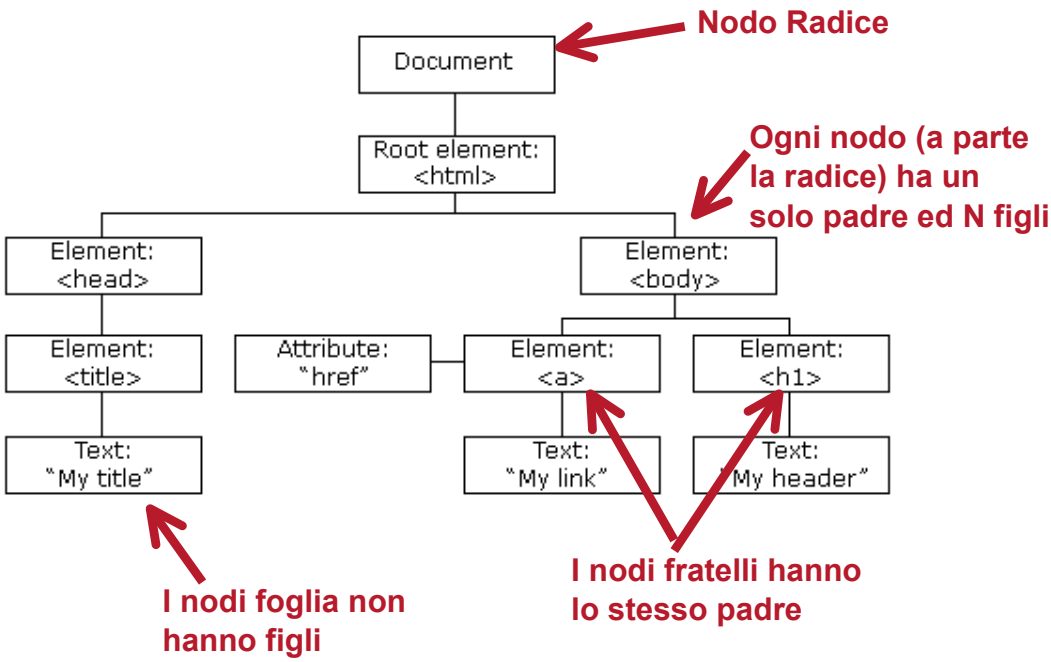
- Il documento HTML è rappresentato da un albero
- Ogni tag HTML è un nodo
- Il testo fra start-tag ed end-tag HTML è sempre rappresentato da un nodo (testo)
- Gli attributi dei tag HTML sono nodi
- I commenti sono nodi

Nota sul Testo

- I nodi che corrispondono a tag HTML non mantengono direttamente il contenuto del testo che delimitano
- Il testo degli elementi e mantenuto in nodi dedicati al testo



Definizioni



Proprietà HTML DOM

- È possibile accedere a varie proprietà dei nodi che rappresentano il documento HTML in DOM
  - X.innerHTML: il valore del testo nell'elemento X
  - X.nodeName: il nome del nodo X
  - X.nodeValue: il valore del nodo X
  - X.parentNode: il nodo padre di X
  - X.childNodes: i nodi figli di X
  - X.attributes: la lista degli attributi di X

Metodi HTML DOM

- È possibile beneficiare di metodi per manipolare l'albero DOM che rappresenta il documento HTML
  - X.getElementByID(*id*): restituisce l'elemento con ID specificato
  - X.getElementByTagName(*name*): restituisce la lista di tutti gli elementi che hanno il tag-name specificato
  - X.appendChild(node): aggiunge un nodo figlio ad X
  - X.removeChild(node): rimuova un nodo figlio da X

Accesso al Contenuto degli Elementi tramite innerHTML

- Sebbene questo metodo non sia parte della specifica DOM del W3C, innerHTML è ampiamente supportato dai browser
- La proprietà **innerHTML** consente di **restituire** o di **modificare** il contenuto degli elementi nel documento HTML
- Può essere anche usato come rudimentale strumento di debug per vedere il sorgente di una pagina modificata dinamicamente

Esempio innerHTML

```
<html>
<body>
<p id="intro">Ciaooo</p>
<p id="altra">altra stringa…</p>

<script type="text/javascript">
tex=document.getElementById("intro").innerHTML;
document.write("il testo è: "+tex);

</body>
</html>
```

Document è il documento HTML, getElementByID restituisce il nodo con l'ID richiesto e innerHTML il testo dell'elemento HTML

La pagina mostrata all'utente contiene

Ciaooo
altra stringa…
il testo è: Ciaooo

Accesso al Contenuto degli Elementi tramite childNodes e nodeValue

- La specifica W3C suggerisce di operare utilizzando:
  - childNodes[i] → restituisce l'i-esimo nodo
  - nodeValue → restituisce il valore di un nodo

```
<html>
<body>
<p id="intro">Ciaooo</p>
<p id="altra">altra stringa…</p>

<script type="text/javascript">
tex=document.getElementById("intro").childNodes[0].nodeValue;
document.write("il testo è: "+tex);

</body>
</html>
```

Restituisce il valore del nodo di testo

Restituisce il primo figlio (childNodes[0]) del nodo trovato con getElementById, ovvero il nodo di testo che mantiene la stringa "Ciaooo"

Proprietà dei Nodi

- I nodi nel modello DOM di HTTP sono oggetti e sono perciò caratterizzati da
  - Metodi
  - Attributi o Proprietà (da non confondersi con gli attributi HTTP)
- I nodi possono avere diverse proprietà fra cui
  - nodeName
  - nodeValue
  - nodeType

nodeName

- Consente di ottenere il nome di un nodo
  - Può essere solo letta
  - Il nodeName di un elemento è il nome del tag associato nel documento HTML (uppercase)
  - Il nodeName di un attributo è il nome dell'attributo
  - Il nodeName di un nodo di testo è #text
  - Il nodeName del nodo document è #document

nodeValue

- Consente di ottenere il valore di un nodo
  - Il nodeValue di un nodo elemento non è definito
  - Il nodeValue di un nodo di testo è il testo in esso contenuto
  - Il nodeValue di un attributo è il suo valore

Esempio: nodeName e nodeValue

```
<html>
<body>
<p id="intro">Esempio</p>
<div id="main">
<p id="main1">I like DOM</p>
<p id="main2">This example demonstrates how to use the
   <b>nodeValue</b> property</p>
</div>
<script type="text/javascript">
x=document.getElementById("intro").firstChild;
document.write("First paragraph text: " + x.nodeValue);
</script>
</body>
</html>
```

Restituisce il riferimento al nodo avente id con valore “intro”

Restituisce il testo contenuto nell’elemento riferito da x

nodeType

- I nodi hanno un tipo. Abbiamo visto infatti come alcuni nodi rappresentino attributi, altri testo, altri elementi, e così via.
- nodeType consente di ottenere il tipo di un nodo
  - Il valore di nodeType può essere solo letto
  - Il valore di nodeType è rappresentato da un numero intero
    - Elemento → 1
    - Attributo → 2
    - Testo → 3
    - Commento → 8
    - Documento → 9

Cambiare il Contenuto degli Elementi

- Cambiare il contenuto del testo

```
<html>
<body>
<p id="p1">Hello World!</p>
<script type="text/javascript">
document.getElementById("p1").innerHTML="New text!";
</script>
</body>
</html>
```

Cambiare il Contenuto degli Elementi

- Il cambiamento potrebbe essere fatto da una funzione handler associata ad un evento

```
<html>
<head>
<script type="text/javascript">
function ChangeText()
   {document.getElementById("p1").innerHTML="New text!";}
</script>
</head>
<body>
<p id="p1">Hello world!</p>
<input type="button" onclick="ChangeText()" value="Premi">
</body>
</html>
```


Cambiare il Contenuto degli Elementi

- Sono possibili molte diverse operazioni
  - Cambiare stile
  - Cambiare colore
  - …
- Sono disponibili librerie per facilitare il lavoro
  - jQuery
  - Mootools
  - …

Oggetti DOM e JavaScript di Interesse

- Window: rappresenta la finestra del browser e viene creata quando il browser incontra <body>
  - Window.history fornisce l'array delle URL associate ai siti visitati dall'utente
- Navigator: restituisce informazioni sul browser
- Screen: fornisce informazioni sullo schermo del client
- Location: fornisce informazioni sulla URL corrente

Ulteriori Oggetti di Interesse

- DOM Document
- DOM Anchor
- DOM Area
- DOM Base
- DOM Body
- DOM Button
- DOM Event
- DOM Form
- DOM Frame
- DOM Frameset
- DOM IFrame
- DOM Image
- DOM Input Button
- DOM Input Checkbox
- DOM Input File
- DOM Input Hidden
- DOM Input Password
- DOM Input Radio
- DOM Input Reset
- DOM Input Submit
- DOM Input Text
- DOM Link
- DOM Meta
- DOM Object
- DOM Option
- DOM Select
- DOM Style
- DOM Table
- DOM TableCell
- DOM TableRow
- DOM Textarea