Programmazione client-side: Javascript

Davide Spano

Università di Cagliari

davide.spano@unica.it

Corso di Amministrazione di Sistema

Javascript

- Fino ad ora, ogni cambiamento nella pagina è sempre corrisposto con un ciclo richiesta/risposta verso il server
- Le applicazioni web odierne però, sono molto dinamiche, nel senso che è sempre possibile mostrare/nascondere delle parti dell'interfaccia
- Inoltre vi sono tante funzionalità comode come suggerimenti, interazioni drag & drop ecc.
- È chiaro che non sia possibile implementare queste funzionalità ricaricando ogni volta la pagina
- È infatti possibile specificare della logica «client-side», cioè programmare il comportamento della nostra pagina in modo che sia dinamica senza la necessità di ricaricarla.
- In queste lezioni vedremo il linguaggio Javascript

Caratteristiche del Javascript

- È un linguaggio di **scripting**, viene cioè utilizzato all'interno di un altro programma per definire dei comportamenti particolari
  - Dentro un browser per definire gli aspetti dinamici dell'interazione utente
  - Ma anche dentro alcuni videogames per definire il comportamento di oggetti e personaggi
- È un linguaggio **interpretato**. Ogni browser esegue un interprete al suo interno ed espone al linguaggio le proprie caratteristiche tramite delle funzioni di libreria
- Ha una sintassi orientata agli oggetti, ma non è un linguaggio OO in senso stretto.

Tag script nella pagina HTML

- Il codice javascript in una pagina HTML può essere aggiunto in due modi
- Includendo il sorgente direttamente nel codice della pagina, tramite il tag **script**
  - ```
    <script type="text/javascript">
      ...
    </script>
    ```
- Collegando la pagina a file esterni, sempre con il tag **script**
  - ```
    <script src="myScript.js"></script>
    ```
- Potete inserire il codice sia nello head che nel body
  - L'esecuzione delle istruzioni non contenute in funzioni avviene man mano che vengono incontrate al caricamento della pagina

Commenti

- Come in altri linguaggi, si riprende la sintassi del C e C++

```
// Commento su linea singola

/*
  Commento su più righe
    ...
*/
```

Istruzioni

- Javascript non ha bisogno dei punti e virgola alla fine di ogni istruzione
  - Nel caso ci sia una sola istruzione per riga
  - Nel caso ne abbiate più di una però si
  - Se ce li mettete anche quando non servono siete sicuri che tutto vada per il verso giusto
- Questo è corretto

```
x += 10
x += 10;
x += 10; y = 7 * 4
```

- Questo non è corretto

```
x += 10 y = 7 *4
```

Variabili

- Al contrario del PHP, non c'è bisogno che il nome dei una variabile inizi con un carattere speciale
- Il primo carattere deve essere per forza una lettera, oppure i simboli `$` o `_` (niente numeri)
- Una variabile può includere sono lettere `a-zA-Z`, numeri `0-9` o i simboli `$` o `_`
- I nomi sono *case-sensitive*, scriverli in lettere minuscole o maiuscole fa differenza
  - `Count count COUNT` sono nomi di variabili diverse

Stringhe

- Le stringhe in javascript si definiscono includendole in apici doppi o singoli
- Non c'è la differenza che avevamo in PHP
  - La stringa viene scritta sempre tale e quale
  - Per scrivere delle parti variabili si usa sempre la concatenazione
- Gli apici singoli o doppi si possono includere nella stringa tramite il carattere di escape \
- ```
  // stringa con doppi apici
  greeting = "Ciao";
  // stringa con apici singoli
  warning = 'Attenzione';
  // concatenazione
  a = 'Un' + greeting + "da \" Cagliari \"";
  ```
  ```

Numeri

- I valori numerici possono essere assegnati alle variabili esattamente come in PHP
- Fate attenzione però alle divisioni ed alle moltiplicazioni, che non si comportano esattamente come in altri linguaggi
- Per esempio l'operatore / per la divisione fra due interi restituisce un numero in virgola mobile
  - In C sarebbe stato un intero
- Esempi

```
x = 5;
x = 9.5;
x = 11 / 2; // 5.5
```

Cambio di tipo

- È possibile trasformare esplicitamente una variabile da un tipo ad un altro

Change to type	Function to use
Int, integer	`parseInt()`
Bool, Boolean	`Boolean()`
Float, double, real	`parseFloat()`
String	`String()`
Array	`split()`

```
n = 3.1415927
i = parseInt(n)
document.write(i)
```

Stampa 3, senza decimali

Dynamic Typing

- Come in PHP, anche in Javascript è possibile che una variabile assuma tipi diversi a seconda del flusso di esecuzione del programma

- Il seguente codice è dunque corretto

```
// qui a e' un intero
a = 27;

if (b > 0) {
    // qui a e' una stringa
    a = "Ciao"
}
```

- Una variabile non inizializzata ha un valore speciale: ***undefined***

Espressioni

- Le espressioni sono simili a quelle del PHP
- Segue la tabella degli operatori Javascript

Operator(s)	Type(s)
() [] .	Parentheses, call, and member
++ --	Increment/decrement
+ - ~ !	Unary, bitwise, and logical
* / %	Arithmetic
+ -	Arithmetic and string
<< >> >>>	Bitwise
< > <= >=	Comparison
== != === !==	Comparison
& ^ |	Bitwise
&&	Logical
||	Logical
? :	Ternary
= += -= *= /= %= <<= >>= >>>= &= ^= |=	Assignment
,	Sequential evaluation

Operatore =, ==, ===

- L'operatore viene utilizzato per l'assegnamento (modifica del valore di una variabile)
- L'operatore == viene utilizzato per comparare due variabili. L'interprete fa del suo meglio per convertire i tipi delle variabili in modo che possano essere comparate
- L'operatore === è l'operatore identità, che verifica che siano uguali sia i valori che i tipi
- ```
  a = 3.1415927
  b = "3.1415927"
  if (a == b)
     document.write("1") // if  vero
  if (a === b)
     document.write("2") // if falso
  ```
  ```

Istruzioni condizionali

```
if (a > 100)
{
    document.write("a is greater than 100");
}
else if(a < 100)
{
    document.write("a is less than 100");
}
else
{
    document.write("a is equal to 100");
}
```

- Non c'è un'istruzione `elseif`

Istruzioni condizionali (2)

```
switch (page)
{
    case "Home": document.write("You selected Home")
        break
    case "About": document.write("You selected About")
        break
    case "News": document.write("You selected News")
        break
    case "Login": document.write("You selected Login")
        break
    case "Links": document.write("You selected Links")
        break
}
```

- Si possono utilizzare le stringhe negli switch come in PHP

Loop

```
//while
counter=0;
while (counter < 5){
    document.write("Counter: " + counter + "<br />");
    counter++;
}

//do-while
count = 1;
do
{
    document.write(count + " times 7 is " + count * 7 + "<br />")'
} while (++count <= 7);


//for
for (count = 1 ; count <= 7 ; ++count)
{
    document.write(count + "times 7 is " + count * 7 + "<br />");
}
```

Funzioni

- Ovviamente è possibile racchiudere del codice all'interno di funzioni
- Le funzioni hanno la seguente sintassi

```
function function_name([parameter [, ...]])
{
    statements
}
```

- Il passaggio di parametri è fatto per **valore**
- Attenzione, gli array funzionano come in Java (non viene copiato l'intero contenuto)
- Si possono restituire valori con la `return`
- L'interprete non si lamenta se non passate il numero giusto di parametri
  - Quelli che non sono passati vengono posti ad ***undefined***
  - Si possono definire funzioni con un numero di parametri non fisso

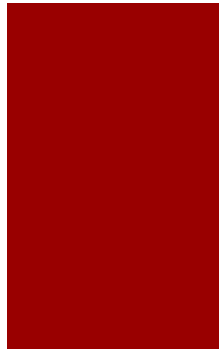
Funzioni (2)

- Le funzioni possono essere assegnate come valore di una variabile
- Le funzioni possono essere restituite come risultato di una computazione
- Le funzioni possono essere passate come parametro
- Sono trattate cioè come tutti gli altri valori del linguaggio di programmazione
  - Numeri
  - Stringhe
  - Booleani
  - Oggetti

Scope delle variabili

- Le variabili in Javascript sono di due tipi
- **Locali** sono visibili solo all'interno della funzione
  - Sono quelle definite utilizzando la parola chiave `var` all'interno di una funzione
- **Globali** sono visibili all'interno di tutte le funzioni e si definiscono in tre modi
  - Tramite istruzioni che non sono contenute all'interno di nessuna funzione
    - Senza la parola chiave `var`
    - Oppure anche con la parola chiave `var`
  - Tramite una definizione di variabile all'interno di una funzione **senza utilizzare la parola chiave** `var`
- Come in PHP, all'interno di una funzione i blocchi non limitano la visibilità delle variabili

Scope delle variabili (2)



```
a = 123 // variabile globale
var b = 456 // variabile globale
if (a == 123) var c = 789 // variabile globale


function test()
{
  a = 123 // variabile globale
  var b = 456 // variabile locale
  if (a == 123) var c = 789 // variabile locale
}
```

Array

- Gli array in javascript sono simili a quelli utilizzati in PHP
  - Si possono indicizzare per posizione
  - Possono essere associativi (vediamo dopo)
  - Implementano diverse strutture dati (liste, code, pile ecc.)
- Una differenza notevole è il fatto che il loro contenuto non venga copiato quando sono passati per parametro ad una funzione
- Si possono costruire delle matrici semplicemente creando degli array di array

Array: esempi

```
// creazione di un array indicizzato per posizione
numbers = Array("One", "Two", "Three");

// aggiunta di un elemento in coda all'array
numbers.push("Four");

// rimozione dell'elemento in posizione 2
numbers.splice(2,1);

// loop su un array
for(var i=0; i<numbers.length; i++){
   document.write(numbers[i]+'<br/>');
}
```

Numero variabile di parametri

```
function displayItems()
{
  for (j = 0 ; j < displayItems.arguments.length ; ++j)
  document.write(displayItems.arguments[j] + "<br />")
}

displayItems("Dog", "Cat", "Pony", "Hamster", "Tortoise");
```

- Questa funzione stampa la lista di argomenti che viene passata
- Come è possibile vedere, gli argomenti non sono dichiarati
- Ma si trovano nell'array `displayItems.arguments`

Array associativi e oggetti

- Come detto in precedenza, in Javascript esistono anche gli array associativi. Sono chiamati *oggetti*
- In realtà, in Javascript tutto ciò che non è un booleano, un numero, una stringa o undefined è un oggetto
  - Quindi un array associativo
- Non esiste un concetto di classe come in PHP o in Java
  - Esistono delle funzioni che costruiscono oggetti con la stessa struttura
  - Ma non ci sono tutte le features della programmazione OO.
  - Es. Non c'è un costrutto del linguaggio per l'ereditarietà
  - È possibile cambiare i metodi di una "istanza" semplicemente assegnando una funzione
- Tuttavia c'è un concetto di classe che raggruppa oggetti che "nascono" con la stessa struttura

For in

- È possibile enumerare tutte le chiavi di un array associativo con il costrutto **for … in**

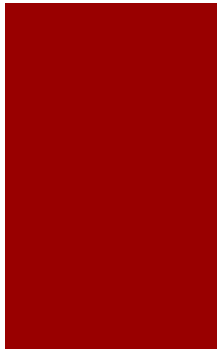
```
var lang = {
    "en" : "Inglese",
    "fr" : "Francese",
    "it" : "Italiano”
};

lang["es"] = "Spagnolo";

for( var l in lang){
  document.writeln("Codice lingua " + l + ": "
      + lang[l] + "<br/>");
}
```

Codice lingua en: Inglese
Codice lingua fr: Francese
Codice lingua it: Italiano
Codice lingua es: Spagnolo

Chiusure lessicali

- In Javascript le funzioni sono *first-class citizens,* cioè possono
  - Essere il valore di una variabile
  - Passate per parametro
  - Restituite come risultato di una computazione
- Una funzione può essere definita quindi all'interno di un'altra
- Ma che succede se riferisce variabili definite nella funzione contenitore e viene poi utilizzata in altri punti del codice?
- Succede che l'insieme delle variabili riferite dalla funzione ed il loro stato viene chiuso (cioè viene mantenuto "vivo") insieme alla funzione stessa
- Questa è la differenza principale fra le funzioni javascript ed i puntatori a funzione del C

Chiusure lessicali: esempio



```
function Counter(){
    // questa variabile continuera' ad
    // esistere anche dopo l'uscita dallo scope
    var count = 1;

    function incrCounter(){
        count = count+1;
        return count;
    }

    return incrCounter;
}

// questa variabile contiene la funzione incrCounter
var cnt = Counter();
// chiamo per la prima volta l'incremento del puntatore,
// la variabile count da 1 passa 2
// da notare che è stata definita nella funzione Counter
// e non nella funzione incrCounter (stampa 2)
document.writeln("[closure]: count, prima chiamata: "+ cnt()+"<br>");
// il valore di count passa da 2 a 3 (stampa 3)
document.writeln("[closure]: count, seconda chiamata: "+ cnt()+"<br>");
```

Chiusure lessicali: funzionamento

```
var cnt = Counter();
```

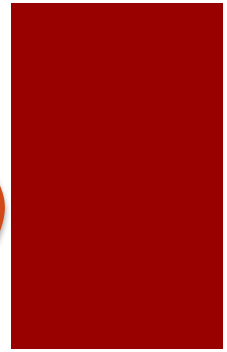
```
var count = 1;

function incrCounter(){
    count = count+1;
    return count;
}

return incrCounter;
```

cnt

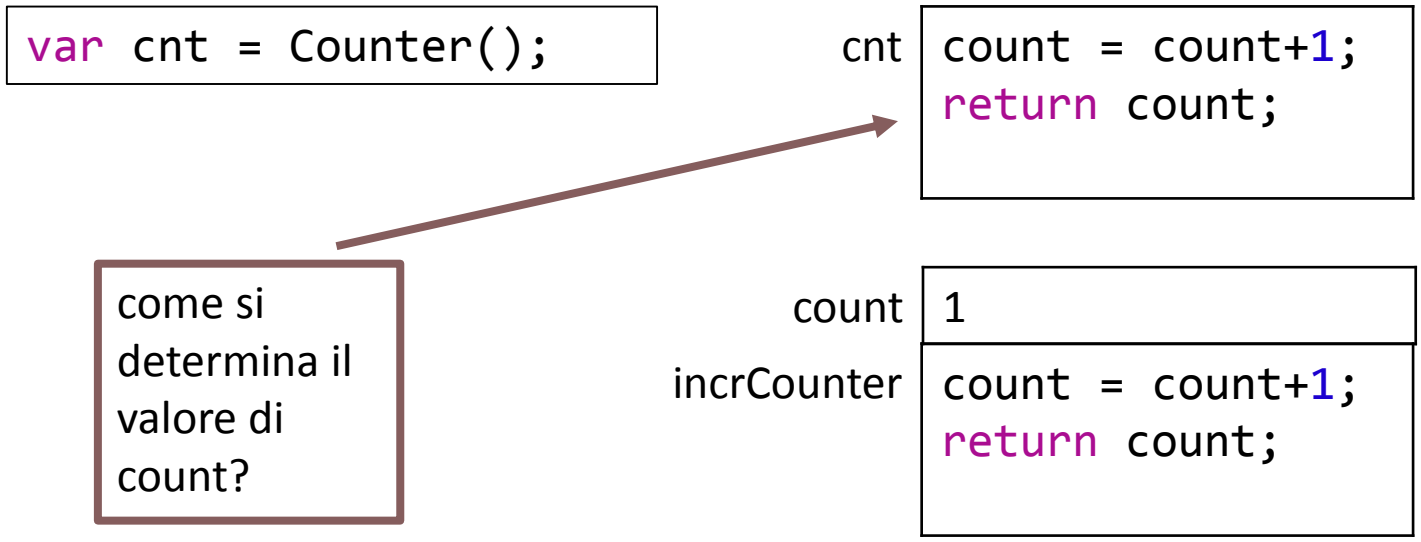
count | 1

incrCounter

```
count = count+1;
return count;
```



Chiusure lessicali: funzionamento



- Questa chiusura viene fatta ogni volta che Counter viene chiamata
  - Quindi ogni volta viene legata una variabile count diversa
  - Che non influenza le altre

Chiusure lessicali: funzionamento

```
var cnt = Counter();
```

cnt

```
count = count+1;
return count;
```

count | 1

```
cnt();
```

cnt

```
count = count+1;
return count;
```

count | ~~1~~ 2

[closure]: count, prima chiamata: 2

Chiusure lessicali: funzionamento

```
cnt();
```

cnt

```
count = count+1;
return count;
```

count ~~2~~ 3

[closure]: count, seconda chiamata: 3

Definizione di oggetti

- Gli oggetti in Javascript possono essere definiti principalmente in due modi:
  - Fornendo una funzione che rappresenta il costruttore degli oggetti di uno stesso tipo (simile ad una classe)
  - Definendone la struttura “al volo” ed assegnandola ad una variabile tramite la short object notation.
  - La seconda tecnica permette di definire un singolo oggetto, si usa cioè quando non è necessario avere più oggetti dello stesso tipo
- Essendo le funzioni dei valori non c’è una distinzione fra variabili e metodi
  - Sono tutti campi dell’oggetto
- È possibile avere qualcosa di equivalente alle variabili statiche aggiungendo dei campi alla funzione che rappresenta il costruttore

Definizione di oggetti (2)

- Il fatto che le funzioni siano dei valori permette di poter modificare quelle che associamo ad un dato oggetto, come per i valori numerici (o altri)
- Perciò è possibile, dopo aver creato un oggetto, cambiare quelli che sono i suoi metodi
  - Semplicemente facendo un assegnamento di variabile
- Nel caso volessimo rendere non modificabili i nostri metodi, li dobbiamo assegnare al campo **prototype** della funzione costruttore
- Il campo prototype è condiviso da **tutti gli oggetti creati tramite quel costruttore**
- Al momento della chiamata, se l'interprete non trova la funzione direttamente nell'oggetto, la ricerca nel prototype
- Variabili e metodi statici possono essere creati utilizzando invece il campo **constructor**

Definizione di oggetti: costruttore

```
function User(forename, username, password){
  // variabili pubbliche
  this.name = forename
  this.username = username
  this.password = password

  // variabile  che tiene il totale degli oggetti
  // creati con questo costruttore
  if(this.constructor.nextId == undefined){
     this.constructor.nextId = 0;
  }

  // variabile privata per l'identificatore unico
  var id = this.constructor.nextId++;
  // questa funzione permette di leggere la variabile
  // privata (nb e' una chiusura lessicale)
  // inoltre, ne viene fatta una copia per ogni oggetto
  this.getId = function(){
     return id;
  }

  // questa funzione e' condivisa tra tutti gli oggetti
  // creati con questo costruttore
  User.prototype.showUser = function(){
    document.write("Nome: " + this.name + "<br />")
    document.write("Username: " + this.username + "<br />")
    document.write("Password: " + this.password + "<br /><br/>")
  }
}
```

Definizione di oggetti: costruttore (2)

```
var usr1 = new User("Davide Spano", "davide", "password");
var usr2 = new User("Pinco Pallino", "pinco", "password2");

// metodi nel prototype
usr1.showUser();
usr2.showUser();

//stampiamo gli identificatori
document.writeln("id davide: " + usr1.getId() + "<br/>");
document.writeln("id pinco: " + usr2.getId() + "<br/>");

// le funzioni incluse nel prototype sono la stessa funzione
document.writeln("usr1.showUser === usr2.showUser ? " +
                 (usr1.showUser === usr2.showUser) + "<br/>");
// le funzioni non incluse nel prototype sono replicate
document.writeln("usr1.getId === usr2.getId ? " +
                 (usr1.getId === usr2.getId) + "<br/>");
// la variabile id non e' accessibile al di fuori del costruttore
document.writeln("valore di usr1.id: " + usr1.id);
```

Nome: Davide Spano
Username: davide
Password: password

Nome: Pinco Pallino
Username: pinco
Password: password2

id davide: 0
id pinco: 1
usr1.showUser === usr2.showUser ? true
usr1.getId === usr2.getId ? false
valore di usr1.id: undefined

Definizione di oggetti: short notation

```
var book = {
    // i nomi delle proprietà possono includere spazi
    "main title": "JavaScript",
    // nel caso si vogliano usare dei caratteri speciali,
    // usare gli apici (singoli o doppi
    'sub-title': "The Definitive Guide",
    "for": "all audiences",
    // si possono avere dei campi che sono a loro volta oggetti
    author: {
        // non e' necessario inserire gli apici nei nomi dei campi
        firstname: "David",
        surname: "Flanagan”
    },
    printName:function(){
        document.writeln('Title: ' + this['main title']);
    }

};
 // come accedere ad un campo che include spazi o altro nel nome?
 // ricordatevi che gli oggetti sono array associativi...
 book['sub-title'];

 // chiamiamo una funzione
 book.printName();
```

Il DOM

- Ma come fa javascript a rendere dinamico l'HTML?
- Fino ad ora abbiamo visto solo che javascript può essere inserito o collegato al codice dalla pagina
- Ora dobbiamo imparare come comunicano
- Vi ricordate la struttura ad albero della pagina HTML?
  - La pagina viene rappresentata come un albero con radice nel nodo HTML
  - Tutti gli altri elementi sono dei nodi dell'albero
  - Viene rappresentata in modo diretto la struttura gerarchica
- Il browser mette a disposizione un DOM (**Document Object Model**), cioè una rappresentazione ad oggetti della struttura ad albero

Oggetti speciali

- Prima di vedere come è fatto il modello ad oggetti della nostra pagina web, vediamo come è possibile accedervi
- Javascript ha un oggetto speciale, gentilmente messo a disposizione da ogni browser
  - NB come per l'HTML alcuni browser (ma in particolar modo uno) hanno delle varianti nell'implementazione di questi oggetti
  - Come vedremo, ci sono delle librerie che ci aiutano…
- Questo oggetto è il **document** che rappresenta il documento correntemente visualizzato
- Esistono vari altri oggetti che ci permettono di interagire con le proprietà del browser, ma noi non li tratteremo a fondo
  - Window
  - Screen
  - History
  - …

Document

- Il **document** è la radice della rappresentazione ad albero della nostra pagina HTML
- Poiché nessun elemento della pagina può esistere senza un documento, il document ha delle funzioni per creare dei nuovi nodi dinamicamente
- Inoltre contiene delle funzioni per ricercare dei nodi in base
  - Al nome del tag
  - All'indentificatore unico del documento
- Le altre funzioni sono comuni a tutti i nodi dell'albero
  - Rappresentato da oggetti di tipo **Node**
  - Elencare i figli
  - Accedere alle caratteristiche del nodo
  - Accedere ai contenuti del nodo
- http://www.w3schools.com/jsref/dom_obj_core_document.asp

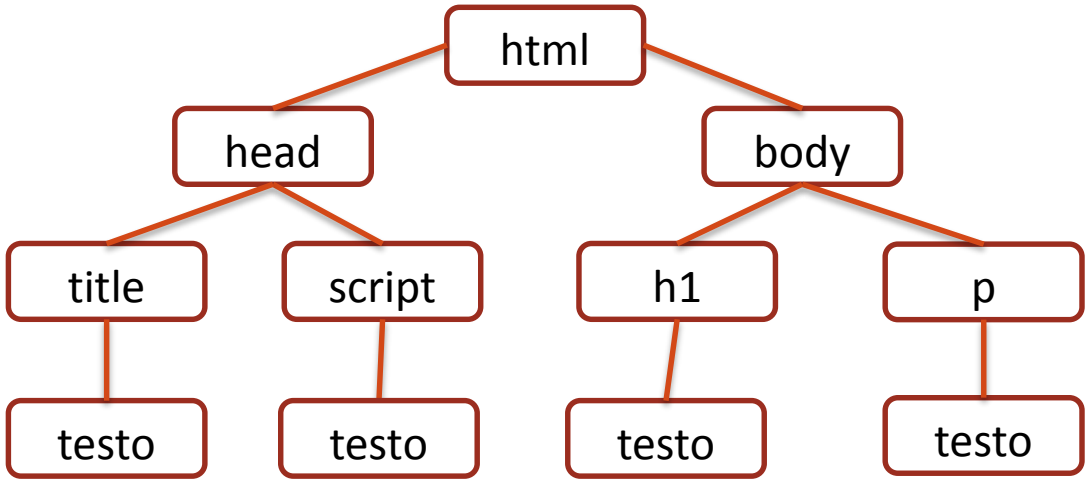
Node

- Rappresenta un nodo nel documento HTML
- Un **nodo** può essere
  - Il documento stesso
  - Un elemento
  - Un attributo
  - Del testo
  - Un commento
- Ha delle funzioni per:
  - Elencare, aggiungere, o eliminare i figli
  - Accedere al nodo padre
  - Elencare gli attributi (se presenti)
  - Accedere/modificarne il valore (es. il testo)
- Proprietà
  - *innerHTML* l'html contenuto all'interno dell'elemento
  - *innerText* il testo contenuto all'intero dell'elemento

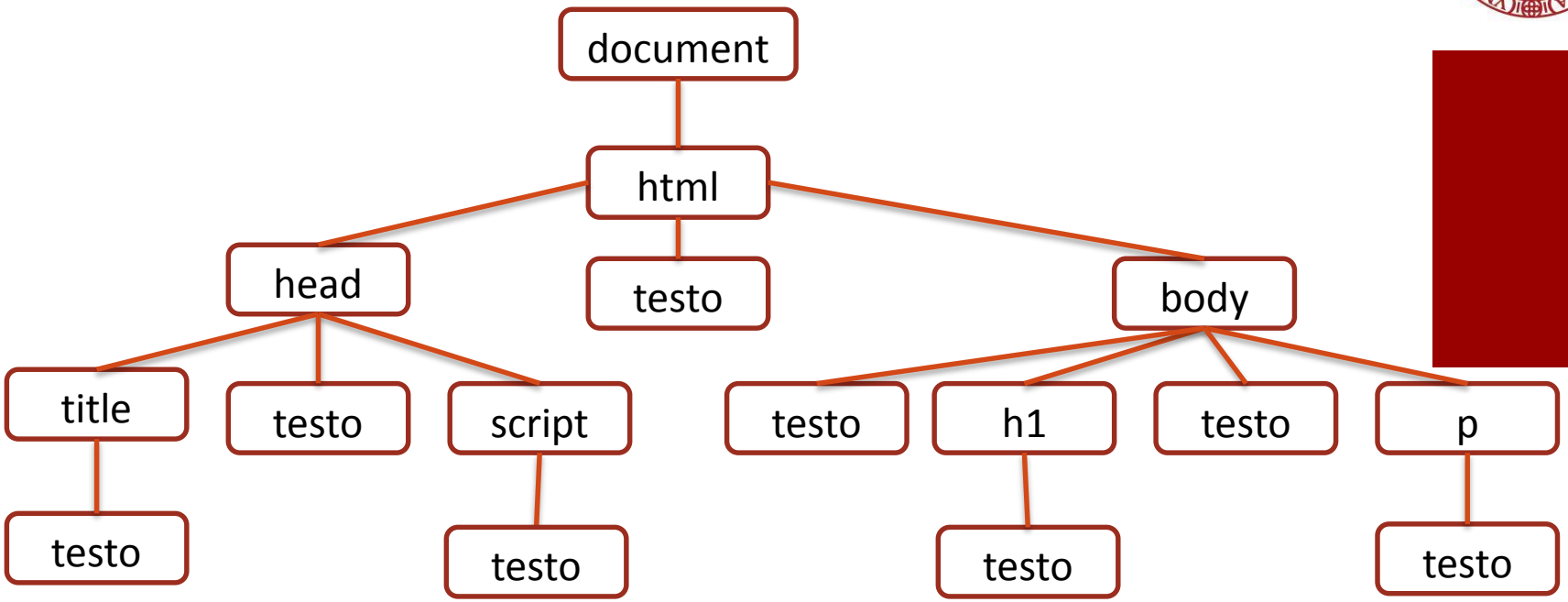
Pagina esempio

```
<!DOCTYPE html>
<html>
<head>
    <title>Prima prova con javascript</title>
    <script type="text/javascript">
        document.getElementById("titolo");
    </script>
    </head>
    <body>
        <h1 id="titolo">Titolo della pagina</h1>
        <p id="paragrafo">Un gran bel paragrafo</p>
    </body>
</html>
```

Struttura ad albero (concettuale)



Struttura ad albero (reale)



Ricerca nel document

- Per identificatore unico

```
function modificaTitolo(){
  // ricerchiamo l'elemento di id titolo
  var titolo = document.getElementById("titolo");
    if(titolo != undefined){
      // abbiamo trovato l'elemento
      // cerchiamo l'elemento testuale
      for(var i in titolo.childNodes){
        var child = titolo.childNodes[i];
        if(child.nodeType == Node.TEXT_NODE){
          // abbiamo trovato il nodo di testo
          // modifichiamo il valore
          child.nodeValue = "Ciao da Javascript”
        }
      }
   }
}
```

Ricerca nel document (2)

- Ricerca per nome del tag

```
function modificaParagrafo(){
   // attenzione questo restituisce tutti i
   // paragrafi nel documento
   var paragrafi = document.getElementsByTagName("p");
   for(var i in paragrafi){
       var paragrafo = paragrafi[i];
       // cerchiamo l'elemento testuale
       for(var j in paragrafo.childNodes){
           var child = paragrafo.childNodes[j];
           if(child.nodeType == Node.TEXT_NODE){
              // abbiamo trovato il nodo di testo
              // modifichiamo il valore
              child.nodeValue = "Ciao da Javascript”
           }
       }

    }

}
```

Aggiunta di nuovi nodi

```
function aggiungiBottone(){
   // creiamo un nodo di tipo button
   var button = document.createElement("button");
   // creiamo un nodo testuale
   var txt = document.createTextNode("Cliccami!");
   // aggiungiamo il nodo testuale al bottone
   button.appendChild(txt);

   // ora aggiungiamo il bottone al body
   document.getElementsByTagName("body")[0].appendChild(button);
}
```

- In modo simmetrico si può eliminare un nodo con la **removeNode**

Modifica CSS

- Ovviamente si possono cambiare non solo i nodi, ma anche le proprietà del CSS
- Possiamo modificare i valori delle proprietà
- Oppure aggiungere e togliere le classi ai diversi elementi
- Ogni nodo di tipo elemento ha un oggetto **style**
  - Attenzione, questo attributo contiene quello che avete definito nell'attributo style
  - La modifica dei CSS è possibile, ma piuttosto difficile
- Però potete aggiungere o rimuovere delle classi ad un elemento tramite l'attributo **className**
  - Che rappresenta esattamente il valore dell'attributo class nell'HTML

Modifica CSS esempi

- Modifica del colore del testo del titolo

```
document.getElementById('titolo')
.style.color ='red';
```

- Aggiunta di una classe CSS

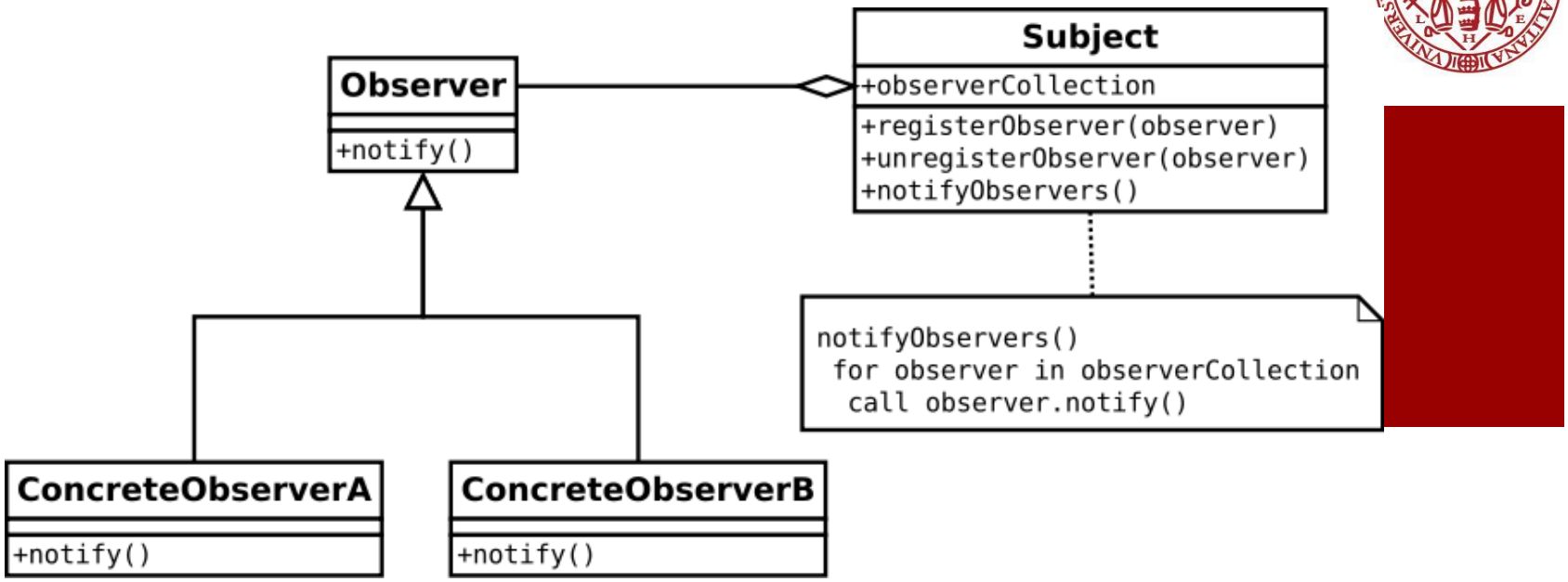
```
document.getElementById('titolo')
.className +=' blueColor';
```

- Attenzione agli spazi quando aggiungete la classe

Gli eventi del documento

- Abbiamo imparato come sono rappresentati i vari elementi del documento
- Abbiamo imparato come modificare le proprietà visuali e non degli elementi
- Ma come facciamo a sapere *quando* dobbiamo effettuare certe operazioni
- Ci aspettiamo per esempio di modificare qualcosa nel documento se l'utente clicca su un bottone, passa sopra un elemento con il mouse ecc.
- Questi sono *eventi* che sono gestibili tramite delle funzioni javascript

Observer pattern



- Abbiamo un “soggetto” che notifica un cambio di stato
- Inoltre abbiamo un insieme di “osservatori” interessati a questo cambio di stato, che implementano tutti un certo metodo (**handler**)
- Il soggetto ha un metodo per registrare un osservatore
- Quando vuole mandare una notifica, invoca il metodo legato all’evento di tutti gli osservatori registrati (notify)
- Soggetto ed osservatore sono completamente disaccopiati

Observer pattern (in pratica)

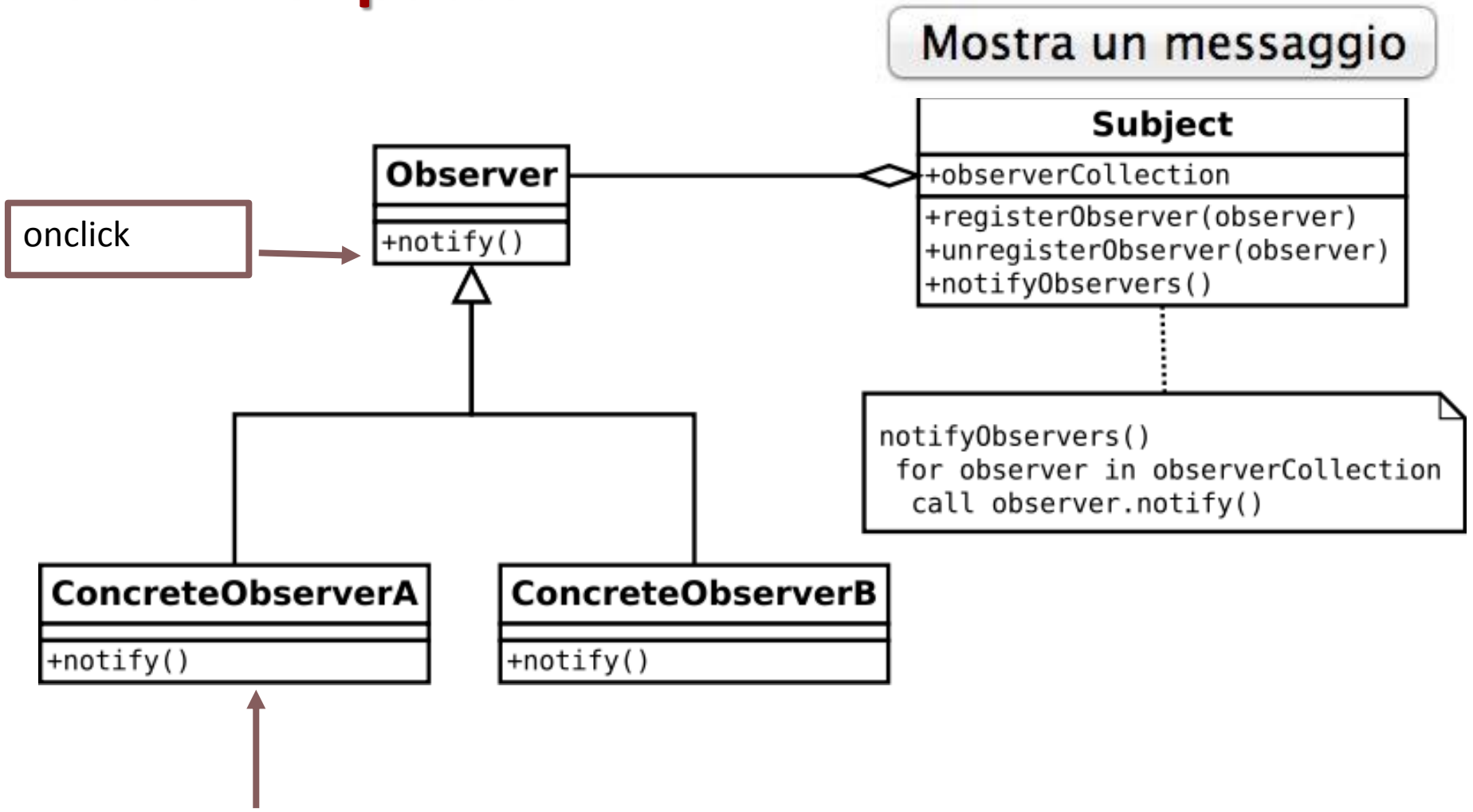
Hai cliccato un bottone!

- Codice javascript da eseguire (**handler**)

```
document.getElementById("par").innerText =
"Hai cliccato un bottone!"
```

- Quando: al click del bottone (**evento**)

Observer pattern

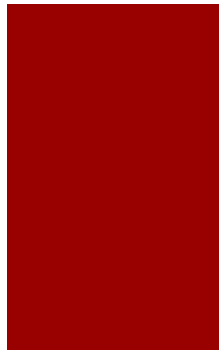



```
document.getElementById("par").innerText =
"Hai cliccato un bottone!"
```

Come agganciare gli handler

- Si può associare un handler per un dato evento specificando una funzione javascript che se ne occupi
- Questa funzione si aggancia all'elemento html tramite specificando il valore corrispondente di un attributo
- Inoltre si può ovviamente assegnare un handler tramite la corrispondente variabile javascript di un elemento
- L'interprete javascript del browser chiama per noi la funzione quando l'evento si scatena
- Tutti gli eventi sono parametrici rispetto ad un **event object** che contiene informazioni addizionali rispetto all'evento
  - Per il mouse la posizione sullo schermo, il bottone premuto ecc.
  - Per la tastiera quale sia il tasto premuto, i tasti modificatori ecc.

Agganciare un handler: esempio



```
function addClick(event){
  //clickCount è una variabile globale
  clickCount++;
  event.currentTarget.innerHTML =
   'Click: ' + clickCount;
}
```

- Direttamente nell'HTML

```
<p id="paragrafo" onclick="addClick(event)">
```

- Via javascript

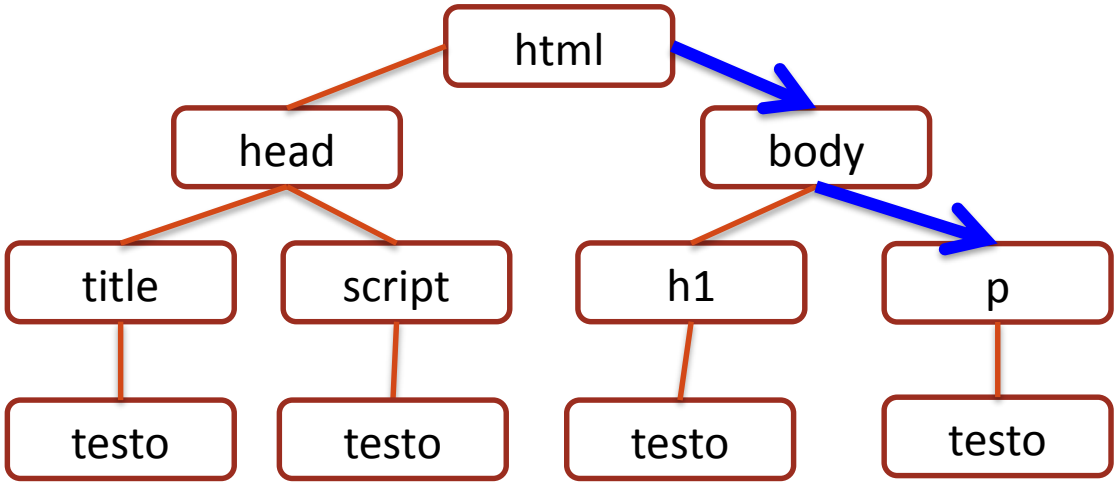
```
document.getElementById('paragrafo').onclick = addClick;
```

Attenzione al this

- CAVEAT: i due modi visti prima **non sono semanticamente equivalenti**
- Dal punto di vista concettuale il primo modo (HTML) è equivalente a scrivere una funzione che contiene l'istruzione che invoca la funzione `addClick`. Questa funzione viene assegnata alla proprietà `onclick` del nostro elemento
- Il secondo assegna la funzione `addClick` ad un campo dell'elemento del DOM, che ne diventa il proprietario quando l'evento si scatena
- Il **this** in javascript rappresenta sempre il "proprietario" della funzione al momento della chiamata
  - Dunque nel primo caso è l'oggetto **window** che rappresenta la finestra del browser (nel caso sia definita in un tag script)
  - Nel secondo caso è l'elemento che rappresenta il paragrafo

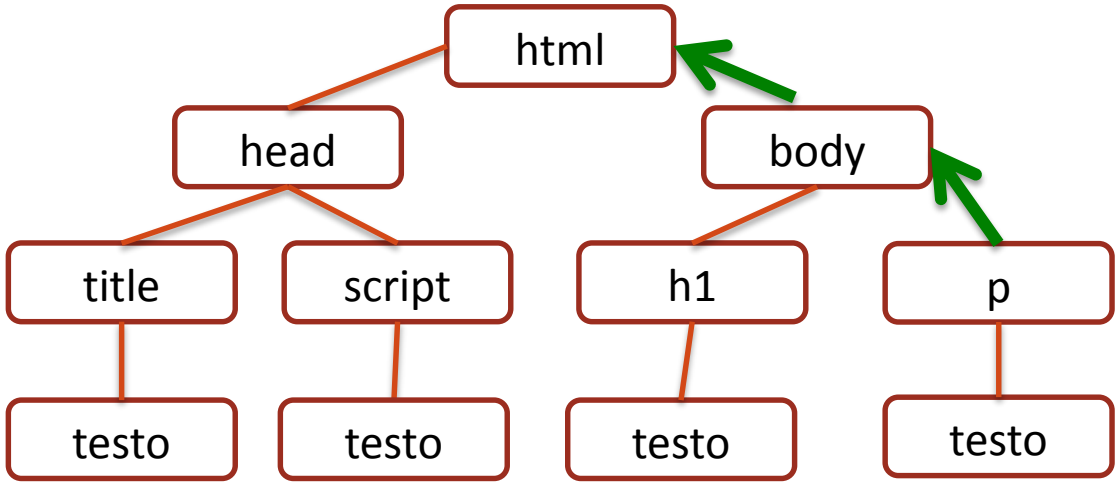
Event tunnelling (click)

- Quando si clicca su un box, il supporto cerca l'elemento più interno che contiene il punto cliccato



Event bubbling (click)

- Dopo di che l'evento "risale" la gerarchia cercando qualcuno che lo gestisca
- Di default viene richiamato ogni handler che si trova



Event target e currentTarget

- All'interno della variabile **event.target** viene mantenuto l'elemento più "profondo" nell'albero dom che ha iniziato il bubbling dell'evento
- Invece in **event.currentTarget** viene mantenuto l'elemento che sta gestendo l'evento (cioè che ha associato l'handler in corso di esecuzione)
- Dunque **event.currentTarget** ed **event.target** possono contenere due valori differenti quando l'evento viene gestito tramite bubbling.
- Es. nell'albero precedente **event.target** è sempre il **p**, mentre il **event.currentTarget** dipende dall'handler

Controllare l'event object

- **event.preventDefault()** fa in modo che le azioni che il browser fa di default non vengano (es. il submit di una form quando si clicca su un bottone di submit)
- In passato, questo effetto veniva gestito semplicemente restituendo **false** alla fine dell'handler
  - Aggiungetelo che non fa male…
- A volte è necessario controllare anche il bubbling degli eventi
- Per esempio se gestiamo il click su un elemento interno e sappiamo che ci può essere anche un handler in un elemento più esterno, può essere utile bloccare il bubbling per evitarne l'esecuzione
- Questo comportamento può essere ottenuto invocando **event.stopPropagation();**

Funzioni e proprietà DOM

- Abbiamo detto che ogni elemento è un nodo
- Ma ovviamente, ha anche delle caratteristiche particolari in base al tipo di elemento
- Queste caratteristiche sono
  - Proprietà
  - Funzioni di controllo specifiche
- Sarebbe troppo lungo elencarle per ogni elemento
- http://www.w3schools.com/jsref/dom_obj_all.asp
  - Trovate tutta la lista nella sidebar sinistra

Proprietà form

- Vediamo come esempio proprietà e funzioni associate ad un form

Form Object Properties

Property	Description
acceptCharset	Sets or returns the value of the accept-charset attribute in a form
action	Sets or returns the value of the action attribute in a form
enctype	Sets or returns the value of the enctype attribute in a form
length	Returns the number of elements in a form
method	Sets or returns the value of the method attribute in a form
name	Sets or returns the value of the name attribute in a form
target	Sets or returns the value of the target attribute in a form

Form Object Methods

Method	Description
reset()	Resets a form
submit()	Submits a form

Eventi del mouse

Mouse Events

Property	Description
onclick	The event occurs when the user clicks on an element
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when a user presses a mouse button over an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseover	The event occurs when the pointer is moved onto an element
onmouseout	The event occurs when a user moves the mouse pointer out of an element
onmouseup	The event occurs when a user releases a mouse button over an element

Eventi della tastiera

Keyboard Events

Attribute	Description
onkeydown	The event occurs when the user is pressing a key
onkeypress	The event occurs when the user presses a key
onkeyup	The event occurs when the user releases a key

Eventi del documento/immagini

Frame/Object Events

Attribute	Description	DOM
onabort	The event occurs when an image is stopped from loading before completely loaded (for <object>)	2
onerror	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)	
onload	The event occurs when a document, frameset, or <object> has been loaded	2
onresize	The event occurs when a document view is resized	2
onscroll	The event occurs when a document view is scrolled	2
onunload	The event occurs once a page has unloaded (for <body> and <frameset>)	2

Eventi dei form/input

Form Events

Attribute	Description
onblur	The event occurs when a form element loses focus
onchange	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
onfocus	The event occurs when an element gets focus (for <label>, <input>, <select>, textarea>, and <button>)
onreset	The event occurs when a form is reset
onselect	The event occurs when a user selects some text (for <input> and <textarea>)
onsubmit	The event occurs when a form is submitted

Proprietà Event Object

Property	Description
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered
button	Returns which mouse button was clicked when an event was triggered
clientX	Returns the horizontal coordinate of the mouse pointer, relative to the current window, when an event was triggered
clientY	Returns the vertical coordinate of the mouse pointer, relative to the current window, when an event was triggered
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered
keyIdentifier	Returns the identifier of a key
keyLocation	Returns the location of the key on the advice
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered
relatedTarget	Returns the element related to the element that triggered the event
screenX	Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered
screenY	Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered

Riferimenti

- Robin Nixon *Learning PHP, MySQL, JavaScript and CSS* O'Reilly,
  - Cap. 13
- David Flanagam: Javascript, the definitive guide
  - Cap 5
  - Cap. 6
  - Cap. 7
  - Cap. 8
- W3C schools
  - http://www.w3schools.com/js/default.asp
  - http://www.w3schools.com/js/js_htmldom.asp