Corso di Web Programming

6. JavaScript Parte I (Aspetti di base)

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea in Informatica Applicata
A.A. 2010/2011

Sommario

Introduzione (1)

- JavaScript è un linguaggio di scripting
  - Di solito per linguaggio di scripting si intende un linguaggio interpretato progettato per svolgere compiti di automazione del sistema o delle applicazioni
- Javascript è IL linguaggio di scripting per il web!
- Il codice JavaScript viene incorporato o allegato ad un documento HTML e viene interpretato dal browser che visualizza la pagina
  - Non viene dato in pasto a nessun compilatore: NO CONTROLLI STATICI (type checking)!
- Tramite JavaScript è possibile conferire dinamicità alle pagine web
  - "animando" i componenti della pagina web
  - modificare il contenuto della pagina web in funzione di input dell'utente
  - effettuando controlli sui dati immessi in un form
  - .....

Introduzione (2)

- JavaScript è stato inizialmente sviluppato dalla Netscape Communications (1995) con il nome di LiveScript e incluso in Netscape Navigator
- Il nome è stato poi cambiato in JavaScript in assonaza con il linguaggio Java, con cui in realtà condivide ben poco
- Successivamente Microsoft ha iniziato a supportare una propria versione JavaScript (detta JScript) in Internet Explorer
- La vita parallela di questi due linguaggi è durata qualche anno, finchè non si è arrivati allo standard ECMAScript
- Il risultato finale è che un linguaggio standard esiste, ma alcune differenze tra le implementazioni dei vari browser non sono ancora state eliminate...

Introduzione (3)

- JavaScript è un linguaggio lato client:
  - Il codice JavaScript viene inviato al client assieme al documento HTML ed eseguito sul computer dell'utente
- Diversamente, linguaggi come PHP sono linguaggi lato server:
  - Il codice PHP viene eseguito sul server e il risultato (un documento HTML) viene inviato al client per la visualizzazione nel browser
- Vantaggi dell'esecuzione lato client:
  - Responsività: durante l'interazione con l'utente non ci sono scambi di dati client-server (più rapidità)
  - Minore carico di lavoro per il server
- Una delle principali limitazioni di JavaScript è la possibilità che esso venga disabilitato dall'utente tramite le impostazioni del browser
  - Per questo è meglio non demandare funzioni importanti (e.g. gestione di dati sensibili) a JavaScript

Introduzione (4)

- Gli errori eventualmente presenti nel codice JavaScript saltano fuori solo a run-time!
  - Non si usano compilatori che effettuano controlli statici (e.g. type checking) sul codice
- I browser di solito mettono a disposizione una console o un logger degli errori (ad esempio Error Console in FireFox) che riporta un messagio quando si verifica un errore
- Esistono anche dei tool di analisi del codice JavaScript, ad esempio il "severissimo" JSLint: `http://www.jslint.com/` utilizzabile via web

JavaScript in HTML

- Per inserire un codice JavaScript in un documento HTML è sufficiente usare il tag `<script>` nel documento HTML
- L'attributo più importante di questo tag è `type`, che specifica il tipo di script usato
  - Il valore nello standard sarebbe `application/x-javascript` ....
  - ... in realtà è molto usato anche `text/javascript` ....
  - ... che nella pratica può anche essere omesso
- Il codice JavaScript può essere incluso nel contenuto del tag `<script>`, oppure si può far riferimento ad un file esterno (solitamente con estensione `.js`) tramite l'attributo `src`

```
<script>
  .... codice JavaScript
</script>
```

```
<script src="mioscript.js">
</script>
```

- Il codice JavaScript può essere inserito sia nella sezione `head` che nella sezione `body` della pagina HTML

Hello World in JavaScript (1)

- Il primo documento HTML+JavaScript:

```
<html>
  <head>
    <title>La mia prima applicazione JavaScript</title>
  </head>
  <body>
    <script>
      document.writeln("Hello, world!");
    </script>
  </body>
</html>
```

- Questo script scrive il testo `Hello, world!` nel documento HTML usando il metodo `writeln` dell'oggetto `document`
  - JavaScript è un linguaggio ad oggetti (un po' particolare)!

Hello World in JavaScript (2)

- Il secondo documento HTML+JavaScript:

```
<html>
  <head>
    <title>La mia prima applicazione JavaScript</title>
  </head>
  <body>
    <script>
      alert("Hello, world!");
    </script>
  </body>
</html>
```

- Questo script apre un alertbox (una finestrella di dialogo) con scritto `Hello, world!`
  - JavaScript consente a un documento HTML di interagire con l'utente!

Hello World in JavaScript (3)

E' bene notare che:

- Il codice JavaScript può comparire in più parti del documento HTML (il tag `<script>` può essere usato più volte all'interno del documento)
- Tutti i frammenti di codice presenti nello stesso documento HTML fanno parte di un unico "programma"
  - una variabile o una funzione definiti all'interno di un tag `<script>` possono essere usate da tutti i frammenti di codice eseguiti successivamente

Associare codice JavaScript a eventi (1)

- E' anche possibile associare l'esecuzione di una porzione di codice JavaScript all'avvenimento di un particolare evento
  - Pressione di un bottone
  - Movimento del mouse su un elemento del documento HTML
  - Modifica dei campi di un form
  - ....
- Esiste un set di eventi che possono essere associati a qualunque elemento del documento html tramite l'uso di attributi.
  - Ad esempio: `<input type="button" onclick="....codice JavaScript....">`
  - Esegue il codice JavaScript quando il bottone viene cliccato (ossia premuto)

Associare codice JavaScript a eventi (2)

Il set di eventi (di solito) include:

onclick	Click sull'elemento
ondblclick	Doppio click sull'elemento
onmousedown	Il tasto (sinistro) del mouse viene premuto
onmouseup	Il tasto (sinistro) del mouse viene rilasciato
onmouseover	Il mouse si trova sopra l'elemento
onmouseout	Il mouse si sposta da sopra l'elemento
onkeypress	Pressione di un tasto della tastiera mentre il focus è sull'elemento
onkeydown	Un tasto della tastiera è premuto mentre il focus è sull'elemento
onkeyup	Un tasto della tastiera è rilasciato mentre il focus è sull'elemento
onfocus	Si pone il focus sull'elemento
onblur	Si sposta il focus su un altro elemento
onchange	L'elemento viene modificato
onsubmit	Si preme il bottone "submit" (per i form)
onreset	Si preme il bottone "reset" (per i form)
onselect	L'utente seleziona il testo contenuto nell'elemento
onload	Al caricamento della pagina/dell'elemento (e.g. body/immagini)
onunload	Alla chiusura della pagina
onresize	La finestra del browser viene ridimensionata

Associare codice JavaScript a eventi (3)

Un altro Hello World che usa gli eventi:

```
<html>
  <head>
    <title>La mia prima applicazione JavaScript</title>
  </head>
  <body>
    <input type="button" value="Clicca qui"
           onclick="alert('Hello, world!')">
  </body>
</html>
```

- Quando si clicca sul bottone "Clicca qui" si apre un alertbox con scritto `Hello, world!`

Associare codice JavaScript a eventi (4)

Con l'approccio a eventi si può facilmente realizzare un *rollover* (un'immagine che cambia quando le si passa sopra il mouse):

```
<img src="logo.png" width="100px"
     onmouseover="this.src='logo2.png'"
     onmousedown="this.src='logo.png'" alt="rollover">
```

- Usando `this` si fa riferimento a un oggetto che rappresenta il tag html corrente
- Gli attributi del tag possono essere acceduti tramite l'oggetto che lo rappresenta (ad esempio `this.src`)

Il linguaggio JavaScript: descrizione generale

JavaScript è un linguaggio di scripting:

- Fondamentalmente imperativo
- Con elementi di programmazione orientata agli oggetti
- Con elementi di programmazione funzionale

Il linguaggio JavaScript: Tipi di Base (1)

- JavaScript è un linguaggio a tipizzazione debole:
  - non c'è bisogno di specificare il tipo delle variabili
  - ogni volta che si usa una variabile (o un valore) essa viene convertita automaticamente nel tipo richiesto dal contesto
- Tra i tipi di dato (impliciti) di JavaScript abbiamo: Numeri (interi e float), Stringhe e Booleani;
- **Numeri:**
  - Per convertire un qualsiasi valore in un valore numerico, JavaScript mette a disposizioni due funzioni `parseInt` e `parseFloat` (e.g. `parseInt("34abc") = 34`)
  - Se il valore passato a `parseInt` e `parseFloat` non può essere convertito in un numero, le funzioni restituiscono NaN (Not a Number), che può essere testato con la funzione `isNaN`

Il linguaggio JavaScript: Tipi di Base (2)

- **Stringhe:**
  - Una stringa può essere rappresentata tra virgolette (e.g. `"ciao"`) o tra apici (e.g. `'ciao'`)
  - Si può usare il carattere di commutazione \ per inserire virgolette o apici all'interno di una stringa
  - Esempi di stringe valide:
    - `"Hello, world"`
    - `'Hello, world'`
    - `"Peter O'Toole"` (ok perchè ' non è stato usato per iniziare la stringa)
    - `'Peter O\'Toole'`
- **Booleani:**
  - Oltre a `true` sono valutati a vero anche
    - Numeri diversi da 0
    - Stringhe non vuote
  - Oltre a `false` sono valutati a false anche
    - Il numero 0
    - La stringa vuota `""` (o `''`)

Il linguaggio JavaScript: Espressioni e Variabili (1)

Gli operatori che si possono usare nelle espressioni JavaScript sono abbastanza standard (simili a C/C++/Java):

- Operatori matematici binari: `+,-,*,/`
- Operatori matematici unari: `-,++,--`
- Operatori stringa: `+` (concatenazione)
- Operatori di confronto: `==,!=,<,>,<=,>=`
- Operatori logici: `&&,||,!`

Inoltre si possono scrivere espressioni condizionali:

- `x>=0?"Positivo":"Negativo"`

Il linguaggio JavaScript: Espressioni e Variabili (2)

- Una variabile si dichiara usando l'istruzione `var`
  - Esempio: `var x;`
  - Esempio: `var y = 125;`
- Non è necessario dichiarare il tipo della variabile (anche perchè non essendo compilato non si fanno controlli statici sul codice)
  - Il tipo della variabile viene determinato ad ogni assegnamento
  - La variabile può cambiare tipo assegnadole valori di tipo diverso in momenti diversi
- Il tipo di una variabile (o espressione) può essere testato usando il comando `typeof`
  - Esempio: `typeof x`
  - Esempio: `typeof (3+5)`
- Il comando `typeof` restituisce una delle seguenti stringhe, a seconda del tipo dell'espressione a cui è applicato:
  - `"number"`, `"string"`, `"boolean"`
  - `"undefined"` (per le variabili dichiarate ma non ancora assegnate)
  - `"object"`
  - `"function"`

Il linguaggio JavaScript: Espressioni e Variabili (3)

- L'operazione di assegnamento è =
  - ma si possono usare anche +=,-=,... come in C/C++/Java
- Le regole di scoping delle variabili sono le solite
- Nelle espressioni i valori (e le variabili) di tipo non compatibile con un'operazione vengono automaticamente convertiti
  - Ad esempio: `"3"-"1"` da come risultato 2
  - **Problema:** l'operazione + è usata sia come somma tra numeri che come concatenzaione di stringhe a seconda del tipo degli operandi!!!

Il linguaggio JavaScript: Espressioni e Variabili (4)

- Bisogna fare molta attenzione quando si usa l'operazione di somma/concatenazione $+$.
- Il seguente frammento di codice JavaScript

```
<script>
  var x=1;
  var y=2;
  document.writeln(x+y+"a");
  document.writeln("a"+x+y);
  document.writeln("a"+(x+y));
</script>
```

produce il seguente output:

```
3a a12 a3
```

Il linguaggio JavaScript: Comandi

Anche i comandi del linguaggio sono come in C/C++/Java:

- Comandi di selezione:
  - `if (trovato) {.....}`
  - `if (trovato) {.....} else {.....}`
  - ```
    switch (msg) {
      case "a" :  ......
                  break;
      case "b" :  ......
                  break;
      default :   ......
    }
    ```
- Comandi di iterazione:
  - `while (!trovato) {.....}`
  - `do {.....} while (!trovato)`
  - `for (var i=0;i<10;i++) {.....}`
    ```

Il linguaggio JavaScript: Array (1)

- Gli array sono presenti in tutti i principali linguaggi di programmazione imperativi/object oriented
- In Java, ad esempio, abbiamo:
  - Gli array veri e propri: dimensione fissata, rapido accesso, sintassi semplice (es.`a[i]`)
  - Classi di libreria (`Vector`, `ArrayList`, ...) che realizzano array dinamici: dimensione variabile, accesso tramite metodi
- In JavaScript abbiamo array dinamici, ma con una sintassi simile a quella degli array statici (es. `a[i]`)

Il linguaggio JavaScript: Array (2)

- Gli array sono istanze della classe Array, quindi un nuovo array si costruisce tramite il comando `new`
  - `var a = new Array();` crea un array vuoto
  - `var a = new Array(5);` crea un array di 5 elementi (non inizializzati)
- In qualunque momento la dimensione (corrente) dell'array può essere testata usando l'espressione `a.length`
- E' possibile anche usare una notazione semplificata per inizializzare gli array:
  - `var a = [];` crea un array vuoto
  - `var a = ["a","e","i","o","u"];` crea un array di 5 elemeni inizializzato;
- La notazione `[....]` può essere usata in qualunque espressione, non solo in fase di dichiarazione di un nuovo array

Il linguaggio JavaScript: Array (3)

- Per accedere a un elemento di un array si usa la solita notazione `a[i]`, con `i` che varia tra 0 e `a.length - 1`
- Come solito, si può accedere agli elementi di un array
  - in lettura (usando `a[i]` dentro un'espressione)
  - o in scrittura (assegnando un valore ad un elemento `a[i]`)
- A differenza del solito, però, gli array sono dinamici!
  - se si accede in scrittura ad un elemento fuori dal range degli indici definiti per un array, l'array viene automaticamente allargato
  - Ad esempio, nel seguente frammento di codice

```
var a = new Array(4);
a[8] = 1000;
```

al momento dell'assegnamento la dimensione dell'array viene automaticamente ampliata da 4 a 9

Il linguaggio JavaScript: Array (4)

- Per effettuare un ciclo che scandisce tutti gli elementi di un array, in JavaScript si può usare la seguente notazione compatta
  - `for (i in a) { ... }`

  che equivale esattamente a
  - `for (var i=0; i<a.length; i++) { ... }`
- Ad esempio, dato una array a costruiamo l'array delle somme parziali:

```
b = new Array(a.length);
for (i in a) {
  b[i] = a[i] + (i>0;b[i-1];0);
}
```

Il linguaggio JavaScript: Funzioni (1)

- E' possibile definire funzioni usando l'istruzione `function`
- La funzione può prevedere parametri di cui non si specifica il tipo
- La funzione può restituire un risultato tramite il comando `return`
- Non è necessario specificare il tipo dei valori eventualmente restituiti dalla funzione

```
function fact(n) {
  if (n=0) return 1;
  else return n * fact(n-1);
}
```

Il linguaggio JavaScript: Funzioni (2)

Elementi di programmazione funzionale in JavaScript:

- E' possibile definire funzioni anonime (omettendo il nome della funzione dopo `function`) da assegnare a variabili o passare ad altre funzioni
- Una funzione anonima è un'espressione, non un comando!!!
- Esempio: la seguente definizione di `fact` è analoga a quella vista in precedenza:

```
var fact = function (n) {
  if (n=0) return 1;
  else return n * fact(n-1);
}
```

segue....

Il linguaggio JavaScript: Funzioni (3)

- Esempio: la seguente funzione `map` applica una funzione `f` ricevuta come parametro a tutti gli elementi di un array `a`:

```
function map(a,f) {
  var b = new Array();
  for (i in a) {
    b[i] = f(a[i]);
  }
  return b;
}
```

- Esempi di uso di `map`:
  `map([1,2,3,4],fact)` restituisce `[1,2,6,24]`
  `map([1,2,3,4],function(n){ return n+1; })` restituisce `[2,3,4,5]`

Il linguaggio JavaScript: Funzioni (3)

Alcune osservazioni sull'uso dei parametri delle funzioni:

- Le variabili di tipo stringa, numerico e booleano vengono passati alle funzioni per valore, mentre gli oggetti vengono passati per riferimento (possono essere modificati dalla funzione che li riceve)
- In realtà non è necessare specificare sempre tutti i parametri quando si chiama una funzione
  - si può chiamare una funzione specificando solo i primi *n* parametri
  - parametri non specificati verranno inizializzati a `null`
- Questo implica che non è possibile fare l'overloading delle funzioni (non si possono definire due funzioni con lo stesso nome)

Il linguaggio JavaScript: Oggetti

- Gli oggetti in JavaScript sono simili alle `struct` di `C`, ma con la possibilità di assegnare funzioni ai vari campi
- Per gli oggetti in JavaScript si usa la seguente terminologia:
  - le variabili di un oggetto sono chiamate proprietà
  - le funzioni di un oggetto sono chiamati (come al solito) metodi
- Un nuovo oggetto si costruisce usando il comando `new` seguito dal costruttore
- Per il momento ci limitiamo a considerare oggetti (e classi) predefinite del linguaggio
  - Vedremo in seguito come definire proprie classi

La classe String (1)

- La classe `String` fornisce metodi per effettuare operazioni sulle stringhe
- Un oggetto della classe `String` può essere creato in due modi:
  - Tramite il costruttore: `var s = new String("una stringa");`
  - In modo automatico, richiamando metodi della classe `String` su una variabile a cui è assegnata una stringa: `var s = "una stringa";`
- Proprietà di un oggetto `s`, istanza della classe `String`:
  - `s.length` – restituisce il numero di caratteri contenuti nella stringa
- Alcuni metodi di un oggetto `s`, istanza della classe `String`:
  - `s.toLowerCase()` – restituisce una nuova stringa ottenuta da `s` mettendo i caratteri in minuscolo
  - `s.toUpperCase()` – restituisce una nuova stringa ottenuta da `s` mettendo i caratteri in maiuscolo

segue....

La classe String (2)

- Metodi di un oggetto `s`, istanza della classe `String`:
  - `s.charAt(n)` – restituisce il carattere della stringa che si trova in posizione `n` (si inizia a contare da `0`)
  - `s.indexOf(s1,n)` – restituisce la posizione della prima occorrenza della sottostringa `s1` all'interno di `s`. Restituisce `-1` se `s1` non è presente. Il parametro `n` (opzionale) specifica la posizione da cui far partire la ricerca
  - `s.lastIndexOf(s1,n)` – analogo al metodo precedente, ma partedo dal fondo della stringa
  - `s.substr(n1,n2)` – restituisce la sottostringa di `s` estratta a partire dalla posizione `n1` per una lunghezza di `n2` caratteri
  - `s.replace(s1,s2)` – restituisce la stringa ottenuta rimpiazzando tutte le occorrenze di `s1` in `s` con `s2`
  - `s.split(sep)` – restituisce un array contenente le diverse sottostringhe in cui la stringa è divisa dal separatore `sep` (a sua volta una stringa, tipo `" "`,`","`,`","`,`"="`,...)

La classe Math

- La classe `Math` fornisce metodi di libreria che eseguono operazioni matematiche comunemente usate nei programmi
- Le proprietà e i metodi della classe `Math` sono statici. Non è necessario creare nuovi oggetti per utilizzarli.
- Alcune proprietà della classe `Math`:
  - `Math.PI` – fornisce il valore (approssimato) di Pi greco
  - `Math.E` – fornisce il valore della costante e (base dei logaritmi naturali)
- Alcuni metodi della classe `Math`:
  - `Math.abs(n)` – restituisce il valore assoluto di `n`
  - `Math.min(n1,n2)` – resituisce il minimo tra `n1` e `n2`
  - `Math.max(n1,n2)` – resituisce il massimo tra `n1` e `n2`
  - `Math.sqrt(n)` – resituisce la radice quadrata di `n`
  - `Math.pow(b,n)` – restituisce il risultato dell'elevamento a potenza b^n
  - `Math.log(n)` – restituisce il logaritmo naturale di `n`
  - `Math.ceil(n)` – restituisce l'arrotondamento per eccesso di `n`
  - `Math.floor(n)` – restituisce l'arrotondamento per difetto di `n`
  - `Math.round(n)` – restituisce l'arrotondamento di `n` (l'intero più vicino)
  - `Math.random()` – restituisce un valore pseudo-casuale tra 0 (incluso) e 1 (escluso)

La classe Date (1)

- La classe `Date` fornisce metodi per gestire date e orari
- Un oggetto della classe date `Date` può essere costruito essenzialmente in due modi:
  - `var d = new Date()` – crea un oggetto inizializzato con la data e l'ora corrente
  - `var d = new Date(anno,mese,giorno,ora,minuti,sec,msec)` – crea un oggetto inizializzato con la data e l'ora passati come parametro.
  - ATTENZIONE: tutti i parametri sono rappresentati come numeri naturali. In particolare `mese` è un numero compreso tra `0` (gen) e `11` (dic)
  - Nell'uso del costruttore più dettagliato non è necessario inserire tutti i parametri (es. `new Date(2011,2,19)` crea un oggetto che rappresenta la data 20 marzo 2010.

La classe Date (2)

- Alcuni metodi di un oggetto `d`, istanza della classe `Date`:
  - `d.getDay()` – restituisce un numero tra 0 e 6 corrispondente al giorno della settimana (0=dom, 1=lun, ecc...)
  - `d.getDate()` – restituisce il giorno del mese
  - `d.getMonth()` – restituisce un numero tra 0 e 11 corrispondente al mese (0=gen, 1=feb, ecc...)
  - `d.getFullYear()` – restituisce l'anno
  - `d.getHours()`, `d.getMinutes()`, `d.getSeconds()`, `d.getMilliseconds()` – restituiscono ore, minuti, secondi e millisecondi, rispettivamente
  - `d.getLocaleString()` – restituisce la data in formato testuale formattata secondo le impostazioni di localizzazione del browser
    - `venerdì 4 marzo 2011` per un italiano
    - `Friday, March 4th, 2011` per un inglese
  - `d.toUTCString()` – restituisce una stringa contente la data e l'ora formattati secondo lo standard UTC (usato spesso):
    - `Fri, 04 Mar 2011 13:24:32 GMT`

segue...

La classe Date (3)

- Alcuni metodi di un oggetto `d`, istanza della classe `Date`:
  - `d.setDate()` – imposta il giorno del mese
  - `d.setMonth()` – imposta un numero tra 0 e 11 corrispondente al mese (0=gen, 1=feb, ecc...)
  - `d.setFullYear()` – imposta l'anno
  - `d.setHours()`, `d.setMinutes()`, `d.setSeconds()`, `d.setMilliseconds()` – imposta ore, minuti, secondi e millisecondi, rispettivamente
- Questi metodi non si limitano a sostituire i valori correnti con i nuovi, ma ricalcolano la data:

```
var d = new Date();
document.writeln(d.toUTCString()+"<br>");
d.setHours(d.getHours()+48);
document.writeln(d.toUTCString());
```

produce il seguente output:

```
Fri, 04 Mar 2011 13:24:32 GMT
Sun, 06 Mar 2011 13:24:32 GMT
```

La classe Date (4)

- La classe `Date` viene spesso usata anche per cronometrare l'esecuzione di un frammento di codice

```
var d1 = new Date();
funzione_da_cronometrare();
var d2 = new Date();
alert("Tempo impiegato: " + (d2-d1) + " millisecondi");
```

Gli oggetti `d2` e `d1` vengono automaticamente convertiti in valori in millisecondi che possono essere confrontati

Oggetto window (1)

- L'oggetto `window` consente di interagire con la finestra del browser
- E' un oggetto globale: se ne posson usare i metodi senza nominarlo
- Le proprietà e i metodi messi a disposizione da `window` possono variare da browser a browser
  - vedremo solo i metodi principali supporatate da tutti i maggiori browser
- Alcuni metodi dell'oggetto `window`:
  - `window.alert(s)` (o `alert(s)`) – apre una finestra di dialogo contenente il messaggio `s` e il tasto `OK`
  - `window.prompt(s)` (o `prompt(s)`) – apre una finestra di dialogo contenente il messaggio `s` e una casella di testo. Restituisce l'input immesso dall'utente nella casella di testo
  - `window.confirm(s)` (o `confirm(s)`) – apre una finestra di dialogo contenente il messaggio `s` e i tasti `OK` e `Annulla`. Restituisce `true` o `false` a seconda del tasto premuto dall'utente
  - `window.open(url,titolo,param)` (o `open(url,titolo,param)`) – apre una nuova finestra (pop-up) con titolo `titolo` e contenente la pagina `url`. In `param` (che non vediamo in dettaglio) si possono specificare proprietà della nuova finestra (ad esempio, le dimensioni)
  - `window.close()` (o `close()`) – chiude la finestra del browser.

Oggetto window (2)

- L'oggetto `window` prevede anche metodi per impostare timeout:
  - `window.setTimeout(s,n)` (o `setTimeout(s,n)`) programma l'esecuzione dell'istruzione scritta nella stringa `s` (ad esempio: `'alert("ciao!")'`) dopo `n` millisecondi e restituisce (immediatamente) un identificatore numerico
  - `window.clearTimeout(id)` (o `clearTimeout(id)`) annulla la programmazione dell'istruzione che aveva restituito l'identificatore `id`
- Esempio: disinnesca la bomba.

```
<script>
var t = setTimeout('alert("BOOM!")',3000);
</script>
<input type="button" onclick="clearTimeout(t)"
       value="disinnesca la bomba">
```

- Esempio: refresh a intervalli regolari.

```
var t = setInterval("location.reload()",2000);
```

Oggetti navigator, location e history

Altri oggetti utili:

- `navigator`: fornisce informazioni utili sulla verisione e sullo stato del browser usato
  - Esempio: `navigator.userAgent()` restituisce l'header HTTP `user-agent` inviato dal browser al server

```
User-agent header sent: Mozilla/5.0 (X11; U; Linux x86_64;
en-US; rv:1.9.2.16) Gecko/20110323 Ubuntu/10.10 (maverick)
Firefox/3.6.16
```

- `location`: fornisce informazioni sull'URL della pagina corrente
  - Esempi: `location.href` fornisce l'URL della pagina corrente; `location.reload()` aggiorna la pagina corrente; `location.assign(url)` apre la pagina specificata da `url`; `location.replace(url)` apre la pagina specificata da `url` senza modificare l'history del browser
- `history`: consente di lavorare con l'history del browser:
  - Esempi: `history.back()` e `history.forward()` consentono di muoversi avanti e indietro tra le pagine nell'history del browser

Oggetto document

- L'oggetto `document` rappresenta il documento HTML corrente
- Consente di accedere al contenuto del documento HTML e di modificarlo (HTML dinamico, vedremo dopo....)
- Alcune proprietà e i metodi di base di `document`:
  - `document.lastModified` – contiene la data e l'ora dell'ultima modifica al documento
  - `document.URL` – contiene l'URL del documento
  - `document.write(s)` – consente di scrivere la stringa `s` (un frammento di HTML) nel documento
  - `document.writeln(s)` – consente di scrivere la stringa `s` (un frammento di HTML) nel documento aggiungendo un newline (nel sorgente HTML generato – per andare a capo nella pagina web bisogna usare `<br>` o simili)

I cookies (1)

- L'oggetto `document` consente anche di accedere ai cookies tramite la proprietà `document.cookie`
- Un cookie è una stringa che ha la forma `nome_cookie = valore; parametri`, dove i parametri di solito riguardano la scadenza del cookie stesso
- La proprietà `document.cookie` contiene una sequenza di stringhe cookies
  - I cookie possono solo essere aggiunti o sovrascritti, ma non rimossi
- Bisogna fare attenzione al formato e all'uso dei caratteri di escape nella stringa dei cookies, quindi:
  - Se si vogliono usare i cookies è bene implementare e usare le seguenti funzioni `setCookie()`, `getCookie()` e `isSetCookie()`

```
function setCookie(c_name,value,exdays)
{
 var exdate=new Date();
 exdate.setDate(exdate.getDate() + exdays);
 var c_exp=((exdays==null) ? "" : "; expires="+exdate.toUTCString());
 var c_value=escape(value);
 document.cookie=c_name + "=" + c_value + c_exp;
}
```

I cookies (2)

```
function getCookie(c_name)
{
var i,x,y,ARRcookies=document.cookie.split(";");
for (i=0;i<ARRcookies.length;i++)
{
  x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));
  y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);
  x=x.replace(/^\s+|\s+$/g,"");
  if (x==c_name)
    {
    return unescape(y);
    }
  }
}
```

```
function isSetCookie(c_name)
{
 var c_value=getCookie(c_name);
 return (c_value!=null && c_value!="");
}
```

I cookies (3)

Un esempio di uso dei cookies:

```
function welcomeback() {
  if (!isSetCookie("utente")) {
    var nomeutente = prompt("Inserisci il tuo nome");
    setCookie("utente",nomeutente,365);
  }
  else {
    var nomeutente = getCookie("utente");
    alert("Bentornato "+nomeutente+"!!!");
  }
}
```

Dynamic HTML (DHTML)

- Con il termine Dynamic HTML (DHTML) si intende un set di tecnologie per realizzare di siti web dinamici che di solito include:
  - HTML
  - CSS
  - JavaScript
  - DOM
- Il Document Object Model (DOM) è uno standard per la rappresentazione di documenti HTML (o XML) in linguaggi orientati agli oggetti
- La specifica del DOM consiste di un insieme di specifiche di interfacce che definiscono le proprietà e i metodi che devono implementare gli oggetti che rappresentano un documento HTML o un singolo tag
- Un documento HTML può essere rappresentato da un albero i cui nodi sono oggetti che implementano le interfacce della specifica DOM
- JavaScript può essere usato per accedere e modificare dinamicamente il documento HTML tramite la sua rappresentazione DOM

Il DOM (1)

- Ogni elemento del DOM di un documento HTML è un oggetto che implementa l'interfaccia `Node` definita dalla specifica dello standard
- La specifica del DOM prevede 12 tipi diversi di nodi, a cui corrispondono implementazioni diverse
- Nella pratica, i tipi di nodo veramente interessanti sono solo 3:
  - il tipo Document: rappresenta l'intero documento (o la sua radice). In JavaScript l'oggetto `document` è un'istanza di questo tipo di nodo
  - il tipo Element: rappresenta un tag del documento
  - il tipo Text: rappresenta un frammento di testo nel contenuto di un tag del documento

Il DOM (2)

- Un nodo di tipo Document (come l'oggetto `document`) fornisce due metodi importanti per accedere agli elementi del documento HTML:
  - `document.getElementById(tag_id)` – restituisce il nodo di tipo Element che corrisponde al tag che ha l'attributo `id` assegnato a `tag_id`
  - `document.getElementsByTagName(tag_name)` – restituisce un array che contiene tutti i nodi di tipo Element che corrispondono ai tag `tag_name` (ad es. `"p"`,`"h1"`,ecc...) nel documento (in ordine)
  - `document.createElement(tag_name)` – restituisce un nuovo oggetto di tipo Element che corrisponde a un nuovo tag `tag_name`
  - `document.createTextNode(s)` – restituisce un nuovo oggetto di tipo Text che corrisponde al frammento di testo contenuto nella stringa `s`

Il DOM (3)

- Un nodo di tipo Element (chiamiamolo `el`, ottenuto ad esempio tramite `document.getElementById()`) fornisce numerosi metodi importanti:
  - `el.getElementById(tag_id)` – analogo al metodo di `document`, ma relativo solo al sotto albero radicato in `el` (ossia la porzione del docuemnto contenuta all'interno del tag che rappresenta `el`)
  - `el.getElementsByTagName(tag_name)` – analogo al metodo di `document`, ma relativo solo al sotto albero radicato in `el` (ossia la porzione del docuemnto contenuta all'interno del tag che rappresenta `el`)
  - `el.firstChild()` e `el.lastChild()` – restituiscono il primo e l'ultimo nodo figlio di `el`, rispettivamente
  - `el.previousSibling()` e `el.nextSibling()` – restituiscono il nodo precedente e successivo rispetto a `el`
  - `el.parentNode()` – restituisce il nodo padre di `el`

....segue

Il DOM (3)

- Un nodo di tipo Element (chiamiamolo `el`, ottenuto ad esempio tramite `document.getElementById()`) fornisce numerosi metodi importanti:
  - `el.getAttribute(attr)` – restituisce il valore di un attriubuto del tag rappresentato da `el` indicato dalla stringa `attr`
  - `el.setAttribute(attr,value)` – assegna all'attriubuto `attr` del tag rappresentato da `el` il valore `value`
  - `el.removeAttribute(attr)` – rimuove l'attriubuto `attr` dal tag rappresentato da `el`
  - `el.appendChild(node)` – inserisce il nodo `node` (di tipo Element o Text) come ultimo figlio del nodo `el`
  - `el.insertNode(node,i)` – inserisce il nodo `node` (di tipo Element o Text) tra i figli del nodo `el` in posizione `i`
  - `el.removeChild(node)` – rimuove il nodo `node` dai figli del nodo `el`
  - `el.replaceChild(node1,node2)` – rimpiazza il nodo `node1` con il nodo `node2` all'interno di `el`
  - `el.hasChildNodes()` – restituisce `true` se il nodo `el` ha figli

Il DOM (4)

Esempio di uso documento HTML dinamico: un contatore di click

```
<html>
  <head>
    <title>Contatore</title>
    <script>
       function incrementa() {
         var i = document.getElementById("i1");
         var v = parseInt(i.getAttribute("value"));
         i.setAttribute("value",v+1);
       }
    </script>
  </head>
  <body>
    <form>
      <input id="i1" type="text" value="0">
      <input type="button" value="incr" onclick="incrementa()">
    </form>
  </body>
</html>
```

Attenzione: modificare l'attributo `value` in alcuni browser potrebbe non risultare in una modifica del contenuto corrente della casella di testo. E' preferibile usare `v.value` (vedremo...)

Il DOM (5)

Esempio di uso documento HTML dinamico: form dinamico

```
<body>
  <form id="myform">
    <input type="submit" value="invia i dati">
  </form>
  <script>
    var n = parseInt(prompt("quanti valori vuoi inserire?"));
    var f = document.getElementById("myform");
    for (var i=0;i<n;i++) {
      var nuovo = document.createElement("input");
      nuovo.setAttribute("type","text");
      nuovo.setAttribute("name","v"+i);
      f.appendChild(nuovo);
    }
  </script>
</body>
```

Il DOM (6)

- E' anche possibile accedere e modificare il contenuto di un tag HTML rappresentato da un oggetto di tipo Element attraverso la proprietà `innerHTML` come nel seguente esempio:

```
<p id="p">Questo paragrafo <strong>scomparir&agrave;</strong>
tra 3 secondi...</p>

<script>
var s = 'document.getElementById("p").innerHTML='
s += '"<strong>Puff!</strong> Sparito!";'
setTimeout(s,3000);
</script>
```