Dynamic HTML grazie al DOM

Far fare qualcosa dinamicamente agli elementi di una pagina web, non è mai stato facile. Tutti conosciamo la difficoltà nello scrivere applicazioni dhtml crossbrowser, ma non tutti forse sappiamo dell'esistenza di una valida soluzione: usare il **Document Object Model (DOM)**. Una delle attivit del W3C [http://www.w3.org/DOM/Activity], ovvero il consorzio che segue lo sviluppo del Web e dei linguaggi da esso usati, stata quella di rilasciare, già a partire dall'ottobre 1998, le specifiche del DOM Level 1 [http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html] ovvero **un'interfaccia standard per la programmazione** (dunque una *API*) **degli elementi di un documento HTML e XML**. Dov'è la "marcia in più" di questa soluzione? Intanto il DOM, descrivendo la struttura di una pagina attraverso i suoi elementi, fa si che ogni programmatore possa interrogarli o manipolarli usando il linguaggio che preferisce (anche se si predilige javascript). Inoltre, come è dovere del W3C, hanno pensato a qualcosa che potesse **funzionare indipendentemente dal sistema operativo e dal browser** in uso! Purtroppo però i produttori di browser non hanno avuto la stessa rapidità nel dotare i loro prodotti di un pieno supporto a questa specifica (ogni browser ha comunque un suo DOM proprietario) anche se, dobbiamo dirlo, le versioni attuali sono decisamente migliorate sotto quest'aspetto.[1] [#nota1] Questo grazie anche alla pressione di progetti come il WASP [http://www.webstandards.org/about/mission/it/].
Ora un po' di teoria: il DOM descrive un modello gerarchico ad oggetti (come un "albero") di tutto ciò che appare in una pagina HTML, e fornisce metodi e attributi per operare su di essi. Dunque un oggetto sar indicato univocamente e riconosciuto dalla sequenza *document.element* e potr essere manipolato grazie ai metodi (le azioni) consentiti dal DOM. Il più importante di questi oggetti è **DOCUMENT** che contiene tutti gli altri. Gli **ELEMENT** sono invece contraddistinti da un tag e possono contenere altri tag o attributi. Infine quando sentiamo parlare di **NODE** vanno intesi tutti gli elementi ma anche attributi, commenti al codice e testo in generale, che invece non possono contenere altri nodi. Dunque tutti gli elementi sono nodi, ma non tutti i nodi sono elementi.
Tutto chiaro? Se no, non preoccupatevi: nel nostro articolo vedremo esempi piuttosto semplici, e poi con esempi pratici le idee spesso si chiariscono.

In questo articolo descriveremo una funzione javascript-dom-CSS molto usata per mostrare o nascondere un elemento (es.1). Poi vedremo come grazie a questa ed altre funzioni si possa simulare una interfaccia visuale a palette, qualcosa di simile a quella adottata dai programmi di grafica (es.2).

La funzione *aprichiudi*

Guardate il codice del nostro esempio 1 [es1.htm]

```
1. function aprichiudi( targetId ){
2.  if (document.getElementById){
3.    target = document.getElementById( targetId );
4.      if (target.style.display == "none"){
5.        target.style.display = "block";
6.      } else {
7.        target.style.display = "none";
8.      }
9.   }
10. }
```

Spiegazione (per riga):

1. Inizio dando alla funzione il nome *aprichiudi* e passandole la variabile che abbiamo chiamato *targetId* (in parentesi, come richiede la sintassi).
2. Con il primo **if** apro un ciclo e controllo se il browser supporta il DOM , in particolare il metodo **getElementById**. Questo metodo mi servirà per recuperare l'elemento che ha per Id il valore unico da me specificato e per potermi riferire ad esso. Se così non fosse, la funzione restituirà **false**, concludendosi.[2] [#nota2]
3. Assegniamo alla variabile *target* il valore di targetId specificato nella chiamata della funzione. (Nell'esempio sarà *box*).
4. L'effetto che desideriamo lo otterremo alternando dinamicamente lo stato dell'attributo "display" (previsto dai CSS) da "block" a "none". Dunque controllo aprendo un nuovo ciclo **if** se il **target.style.display** (ovvero l'attributo di stile *display* dell'elemento *target*) è uguale a *none*.
5. se ciò e vero, significa che il nostro elemento (*box*) è nascosto. Quindi devo renderlo visibile cambiando target.style.display in ***block***.
6. se era falso, eseguo il codice dopo l'**else**
7. il box è quindi visibile e devo nasconderlo portando target.style.display a ***none***.

Per funzionare occorre solo che nell'HTML sia presente:

- L' elemento univoco che vogliamo aprire/chiudere, con il suo id e con l'attributo display impostato. Se di default metttiamo *none*, l'elemento al caricamento della pagina non viene mostrato, anzi per l'esattezza è come se non esistesse nemmeno in quanto non genera alcun box (se volessimo il contrario, ovvero che in partenza il box sia visibile, possiamo farlo mettendo come display *block*). L'importante è **specificare sempre tramite l'uso di CSS in linea uno dei due valori** e non con un CSS importato o linkato!

```
<div id="box" style="display:none">
```

```
        testo di prova che verrà mostrato o nascosto
        ogni volta che si clicca sul link aprichiudi
      </div>
```

- Un link che richiami la funzione e le passi il valore da attribuire a *targetId*:
  `<a href="#" onclick="aprichiudi('box')">mostra/nascondi il box</a>`

Trascinare il box

Il nostro obiettivo è quello di arrivare a simulare le palette dei programmi di grafica. Dunque il prossimo passo sarà quello di permettere al box di essere trascinato per lo schermo. Ci servir dotarlo di una "maniglia". Per far ciò useremo lo script DOM-drag [http://www.youngpup.net/?request=/components/dom-drag.xml] creato da Youngpup.net . Non è per obiettivo del nostro articolo spiegarne il funzionamento, tral'altro ben descritto dai tutorial preparati dal suo inventore. Noi vedremo solo come "usarlo" per il nostro scopo, esattamente come abbiamo fatto nell'esempio 2 [es2.htm].
Intanto linkiamo il file esterno (dom-drag.jsp) contenente lo script:

```
<script language="javascript" src="dom-drag.js"></script>
```

Ora occorre configurare le variabili necessarie per utilizzare questo script nel nostro esempio:

```
1. var theHandle, theRoot;

2.  window.onload = function() {
3.    theHandle = document.getElementById("maniglia");
4.    theRoot   = document.getElementById("paletta");
5.    theRoot.style.left = "50px";
6.    theRoot.style.top  = "50px";

7.    Drag.init(theHandle, theRoot);
8.  }
```

Spiegazione:

1. Dichiaro le due variabili che useremo: *theHandle* e *theRoot*
2. Lancio al caricamento della pagina la funzione che:
3. associa alla variabile *theHandle* l'elemento a cui ho dato come ID *maniglia*
4. associa a *theRoot* il mio ID *paletta*

5. Ora posiziono nella pagina *theRoot* (che nel mio esempio contiene come div annidato anche *theHandle*) grazie ai CSS e al posizionamento assoluto. Decido per 50 px a sinistra (CSS prende come riferimento l'angolo alto destra)
6. e 50 px dall'alto
7. Solo ora posso chiamare la funzione **drag.init()** contenuta nello script esterno che serve a rendere i miei oggetti "trasportabili".

Siamo invece liberi di dare ai nostri elementi paletta, maniglia e box l'aspetto che vogliamo sia usando i CSS che con eventuali immagini. Anche nell'HTML godiamo di buona libertà tanto da poter creare molteplici varianti di questa applicazione.
Ecco il nostro esempio finale [es_finale.htm]. Ora tocca a voi!

Note:

[1] Internet Explorer e Netscape superiori alla versione 4, Opera dopo la 6, e tutti i browser Gecko-based (Mozilla, Galeon ecc.ecc.) supportano pienamente il DOM Level1 e "quasi pienamente" il Level2. [si vedano gli "Approfondimenti"]Nonostante ciò DOM va usato con attenzione, la stessa di sempre quando si ha a che fare con DHTML. Se non siete certi che tutti i vostri utenti usino strumenti aggiornati per navigare, ricordate di fornire loro una "soluzione alternativa" come prevedono le norme di accessibilit.

[2] Eseguiamo un controllo per verificare se il browser dell'utente supporta il DOM e il metodo che useremo; ovviamente in caso contrario la nostra funzione non produrrà alcun effetto. ("getElementById" funziona in tutti i browser superiori alla versione 4).

Approfondimenti (link a siti esterni):

- Ora è in sviluppo il DOM Level3 [http://www.w3.org/DOM/]
- W3C Standards Support in IE and the Netscape Gecko Browser Engine [http://wp.netscape.com/browsers/future/standards.html]
- Microsoft About the W3C Document Object Model [http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dom/domoverview.asp]
- Supporto delle specifiche in Opera6 [http://www.opera.com/docs/specs/opera6/index.dml] e Opera7 [http://www.opera.com/docs/specs/index.dml]
- Il DOM su QuirksMode.org [http://www.quirksmode.org/?dom/contents.html] di Peter-Paul Koch

Christian Fusi | febbraio 2004