JavaScript: jQuery sì o no?

Una delle funzionalità più apprezzate di JavaScript è senza dubbio la sua facile estendibilità: questo linguaggio di programmazione permette infatti di creare plugin e/o framework con grande semplicità. Fra questi c'è senz'altro jQuery.

La facilità di utilizzo di jQuery lo fa preferire sovente a JavaScript nativo, in quanto dispone di moltissimi metodi che spaziano in tutti gli ambiti: a partire da AJAX (come già spiegato in questo articolo), fino ad arrivare all'iterazione senza utilizzare cicli iterativi diretti quali *for* o *while*.

Tuttavia, l'utilizzo di jQuery **non è sempre consigliato**. Spesso ci sono funzioni che possiamo ugualmente ottenere senza ricorrere a plugin o framework, ma semplicemente con un'aggiunta di pochissimo olio di gomito, visto che su progetti di larghe dimensioni anche un piccolo script di 27kb potrebbe avere un pesante impatto sulle prestazioni generali.

A questo proposito, ti elencherò di seguito una serie di 5 piccoli tip che potrai utilizzare e riadattare per qualsiasi tuo lavoro, senza il bisogno di scomodare CDN vari. Naturalmente, i seguenti consigli valgono anche per qualsiasi altro framework o plugin.

1 - Il wrapper "$"

Capita alle volte di utilizzare jQuery esclusivamente per la **comodità dei suoi selettori** e, in particolare, del suo wrapper "$". Una motivazione abbastanza debole, in quanto in JavaScript lo stesso effetto è facilmente ottenibile tramite il selettore *querySelectorAll*. Vediamone un breve esempio qui sotto.

Se questo procedimento ti sembra troppo lungo, puoi rimediare con la creazione di una semplice funzione che vada a sostituire il wrapper.

Potrai poi utilizzarla nel seguente modo:

Il metodo querySelectorAll necessita, tuttavia, di una successiva iterazione, in quanto il suo risultato è una lista di nodi (*NodeList*) — ovvero elementi — che bisogna processare per poter manipolare i singoli nodi contenuti all'interno. Se ti sembra complesso, non preoccuparti, vedrai come si può facilmente ovviare a ciò nel punto successivo.

2 - L'iterazione con le NodeList

jQuery è famoso per permettere una manipolazione estremamente malleabile dei suoi elementi tenendo gran parte dei processi nascosti allo sviluppatore. Ciò non è sempre un bene, in quanto non si ha mai l'esatto controllo del funzionamento dello script.

Capire le NodeList è davvero semplice: basti sapere che esse non sono nient'altro che array. Sarà spontaneo il ragionamento di iterare attraverso i diversi nodi tramite un semplice ciclo *for*.

Con questa semplicissima porzione di codice, per esempio potremo assegnare un'altezza di 200px a tutti gli elementi presenti nella NodeList.

3 - Gli eventi

Chiunque abbia provato jQuery avrà sicuramente provato anche i suoi fantastici eventi. O forse **non sono così meravigliosi** come sembrano? Scopriamolo nel codice appena sotto.

Come puoi notare, stavolta non abbiamo utilizzato la funzione *S* precedentemente creata, ma abbiamo impiegato il selettore *querySelector* anziché querySelectorAll; questo perché non ci occorre selezionare più elementi, visto che in questo caso ce ne basta solo uno.

Se hai fatto attenzione, avrai sicuramente notato che non iteriamo l'elemento in un ciclo for, dato che il selettore *querySelector* restituisce un solo nodo al posto di una NodeList.

4 - La gestione delle classi

Un grande pregio di jQuery è quello di offrire ottimi metodi per la gestione **on the fly** delle classi. Tuttavia JavaScript offre dei metodi ugualmente potenti, seppur poco conosciuti:

Hai capito bene: i metodi *.add*, *.remove* e *.toggle* sono proprio l'esatto equivalente di quelli che utilizzi con jQuery. Semplice, no?

5 - Le animazioni

Arriviamo ora al tasto dolente di jQuery. Purtroppo questo framework non offre un gran metodo per le animazioni, dato che non permette ancora di animare una proprietà quale *transform*.

In questo punto ti chiedo una particolare attenzione, visto che il codice non è esattamente breve, ma una volta appresi i concetti di base, potrai realizzare soluzioni ad **alto impatto visivo**. Ecco lo snippet qui di seguito:

Innanzitutto assegniamo una variabile al nostro elemento *another-div*. Successivamente, stabiliamo il numero di pixel entro il quale la nostra animazione deve terminare e salviamo tale valore nella variabile *endValue*.

Fatto ciò, creiamo la funzione *translateX()* e assegniamo a una variabile il primo valore che dovrà avere la proprietà che andremo a modificare in seguito, che nel nostro caso sarà pari a zero. Successivamente definiamo un'altra funzione interna a quella attuale, *incrementTranslate()*, che verrà successivamente chiamata ogni ms grazie al *setInterval* definito poco dopo; così facendo si andrà a incrementare la variabile ogni qual volta la funzione verrà chiamata, attribuendo alle nostre proprietà di volta in volta il suddetto incremento.

Siamo quasi giunti al termine: quando la variabile *current* sarà uguale ad *endValue*, l'elemento avrà eseguito correttamente la nostra animazione, motivo per cui andremo a cancellare il *setInterval* settato in precedenza, mettendo fine così all'animazione in essere.

Infine, al caricamento degli elementi — *window.onload* è l'equivalente JavaScript di *$(document).ready()* — avvieremo semplicemente l'animazione dell'elemento.

Conclusioni

Anche oggi l'articolo termina qui. Se adesso stai per gettare nel cestino i tuoi script pieni di $ e .toggleClass(), non farlo: jQuery è uno dei framework JavaScript più validi in circolazione; offre tool comodissimi nello sviluppo di siti medio-grandi, senza contare la vastissima disponibilità di plugin aggiuntivi e il supporto che riceve da una massiccia comunità internazionale.

Lo scopo di questo articolo **non è quello di convertirti** alla dottrina del "*Pure JavaScript*", ma vuole lasciarti con una domanda: "Ho sempre bisogno di ciò che utilizzo?".

Se ti va, facci sapere cosa ne pensi a proposito; sono fermamente convinto che potrebbero nascere spunti più che interessanti. Per ora ti auguro come al solito una buona giornata e un buon coding!