JQUERY

Introduzione

* di [Marco Solazzi](https://www.html.it/author/marcosol/)
* 29 giugno 2009

A guardare bene, sotto lo strato della socialità, dell’interattività e dello *User Generated Content*, il web 2.0 con il quale tutti noi abbiamo a che fare giornalmente è basato massicciamente sull’uso di JavaScript. In passato spesso associato all’idea di popup invasivi e fraudolenti, questo linguaggio è stato particolarmente rivalutato negli ultimi anni anche grazie alla diffusione di tecniche come AJAX e alla possibilità sempre più estesa di riprodurre effetti grafici dinamici senza la necessità di plugin come Flash.

Dal punto di vista degli sviluppatori, tuttavia, la natura *lato client* del linguaggio è da sempre la fonte principale di problemi e mal di testa. Infatti, poiché ogni browser implementa uno specifico motore JavaScript, ognuno (ma soprattutto IE) con specifiche ed eccezioni proprie, è spesso impossibile essere certi del funzionamento cross-browser di uno script. Inoltre, il linguaggio presenta ancora notevoli lacune che lo rendono incoerente rispetto ai linguaggi server-side come PHP o Ruby.

In risposta a questi problemi sono nati progetti di librerie (o meglio *framework*) in grado di garantire il funzionamento cross-browser degli script e di estendere o comunque facilitare le funzioni native di JavaScript.

Uno dei primi framework a vedere la luce è stato **[Prototype](http://www.prototypejs.org/" \t "_blank)**, che ha approcciato il problema estendendo il DOM e gli oggetti globali di JavaScript, cioè aggiungendo funzionalità agli elementi che compongono la pagina e ad oggetti come array e strighe. Con questo metodo vengono colmate le numerose lacune di JavaScript allo stesso modo in cui vengono aggiunti plugin in vari programmi. Nelle sue prime versioni anche **[Mootools](http://mootools.net/" \t "_blank)**aveva intrapreso questa strada, mentre nell’ultima versione la libreria è quasi completamente strutturata ad oggetti, mantenendo l’estensione di alcuni oggetti fondamentali come array e stringhe. Il problema dell’approccio di Prototype è infatti che estendendo gli oggetti globali di JavaScript si potrebbero creare **incompatibilità con nuove versioni dei motori**o addirittura conflitti con le funzioni dello sviluppatore o con altre librerie.

**Il progetto jQuery**

Un approccio molto diverso è da sempre quello di **[jQuery](http://jquery.com/" \t "_blank)**, un framework sviluppato da John Resig a partire dal 2006 con il preciso intento di rendere il codice più sintetico e di limitare al minimo l’estensione degli oggetti globali per ottenere la massima compatibilità con altre librerie.

Da questo principio è nata una libreria in grado di offrire un’ampia gamma di funzionalità, che vanno dalla manipolazione degli stili CSS e degli elementi HTML, agli effetti grafici per passare a comodi metodi per chiamate AJAX cross-browser. Il tutto, appunto, senza toccare nessuno degli oggetti nativi JavaScript.

Per chi si avvicina a jQuery per la prima volta è quindi fondamentale sapere che tutto ruota attorno all’oggetto/funzione $, a sua volta un abbreviazione (o alias) di jQuery.

$("#mioBlocco"); //Un oggetto jQuery  
jQuery("#mioBlocco") //Lo stesso oggetto richiamato in modo diverso

Ma andiamo con ordine, ecco un primo esempio per avere un colpo d’occhio su vantaggi di jQuery:

// trovare il valore dell'attributo href del tag a con id mioLink

// <a id="mioLink" href="http://www.html.it">link</a>

document.getElementById("mioLink").href; // JavaScript nativo  
$('mioLink').readAttribute('href'); // Prototype  
document.getElementById("mioLink").readAttribute('href') // Prototype, come detto, estende il DOM  
$("#mioLink").attr("href"); // jQuery

Già da queste righe di codice si intuiscono due caratteristiche fondamentali di jQuery: anzitutto la **brevità del codice utilizzato**, ma soprattutto il fatto che l’elemento da ricercare sia stato passato alla funzione $() sotto forma di selettore CSS (come avviene in Mootools per la funzione $$ ).

Per chi non ha dimestichezza con framework del genere, o proviene da altre librerie, inizialmente questa caratteristica potrebbe sembrare uno svantaggio e magari produrrà qualche errore: per esempio in jQuery la funzione $('mioLink') cercherà un immaginario tag mioLink senza trovarlo.

In realtà uno dei punti di forza del framework è proprio il potente e performante motore di selezione (di cui parleremo più avanti).

Un’altra caratteristica che va sottolineata è che molti metodi di jQuery sono **concatenabili**, per rendere la scrittura e la lettura del codice molto più lineare:

$("#mioLink").text("Nuovo testo").css("color","red");   
//cambio il testo del link e il colore

[**Esempio**](http://www.html.it/guide/esempi/jquery/esempi/introduzione.html).

**I buoni motivi per usare jQuery**

Oltre ai vantaggi già citati, vi sono alcuni buoni motivi per cui è una buona idea usare jQuery: anzitutto perché è **possibile usarla in tutti progetti** senza paura di incappare in incompatibilità nel codice. Infatti, utilizzando la funzione jQuery.noConflict() verrà semplicemente rimosso l’alias $, e potremo usare, per esempio, Mootools richiamando questo framewok con $('mioId') e jQuery con jQuery("#mioId")([**esempio**](http://www.html.it/guide/esempi/jquery/esempi/introduzione2.html)).

In secondo luogo, jQuery ha un semplice **sistema di estensione**che permette di aggiungere nuove funzionalità (plugin) oltre a quelle predefinite.

Inoltre la sua diffusione ha fatto in modo che attorno al team di sviluppo ufficiale crescesse una **numerosa community** che mette a disposizione plugin e supporto di ottimo livello.

Infine, e potrebbe sembrare una cosa da poco, perché il motto di jQuery è *“Write less, do more”*, ed effettivamente la sua **sintassi sintetica ed efficiente** è particolarmente apprezzabile quando si tratta di scrivere svariate linee di codice.

**Ottenere jQuery**

Dal sito [**jquery.com**](http://jquery.com/) è possibile scaricare il framework in due formati:

* **development**: non compresso, utile in fase di sviluppo e debug (120Kb)
* **production**: compresso, da utilizzare quando il progetto è online per ottimizzare i tempi di caricamento (59Kb, 19Kb se compresso ulteriormente lato server con GZip)

Una volta scaricato il file, sarà sufficiente collegarlo alla pagina HTML inserendo nel tag head il seguente codice:

<script language="javascript" type="text/javascript" src="jquery-1.3.2.min.js"></script>

Un’altra soluzione che si è fatta strada nell’ultimo periodo, è quella di utilizzare una copia del framework presente sul**network di Google**:

<script language="javascript" type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>

Questa soluzione lega il funzionamento dei nostri script alla raggiungibilità del servizio (mai garantita al 100%) ma ha il vantaggio che l’utente potrebbe già aver scaricato il file in questione nella cache del browser e per questo gli verrebbe evitato di dover riscaricare un file uguale con conseguente **diminuzione del tempo di caricamento della pagina**.

# Per un pugno di $: il motore di jQuery

* di [Marco Solazzi](https://www.html.it/author/marcosol/)
* 29 giugno 2009

Come detto in precedenza tutto ciò che possiamo fare con jQuery gira intorno all’oggetto/funzione $, il cui utilizzo più comune è quello di selettore CSS:

var elementi = $("#mioId");

In questo esempio la variabile elementi rappresenta un oggetto jQuery contentente un riferimento all’elemento con id mioId.  
Come detto in precedenza la stringa passata alla funzione deve rappresentare il**selettore CSS** **degli elementi da ricercare**, siano essi un singolo id oppure elementi accomunati da una classe:

$(".miaClasse"); //tutti gli elementi con attributo class "miaClasse"

Come in un foglio di stile sarà possibile indicare **più di un selettore** anche con sintassi piuttosto complesse:

00:10

/

02:30

00:00

**Share**

**Playlist**

**Link**



*Copy to clipboard: Ctrl+C / Cmd+C*

**Embed**



*Copy to clipboard: Ctrl+C / Cmd+C*

$(".miaClasse, #mioId, ul#mioMenu li");  
//cerca una classe, un id e gli elementi di una specifica lista puntata

Vedremo più avanti le possibili combinazioni di selettori CSS ricercabili con jQuery, ma per completezza è importante dire che, nel caso di ricerca con selettori CSS, è possibile passare alla funzione un secondo argomento per indicare un contesto nel quale operare:

$(".miaClasse",document.getElementById('mioId'));  
//Tutti gli elementi con classe "miaClasse" dentro l'elemento DOM con id mioId

Questa caratteristica sarà molto utile quando dovremo ciclare fra gli elementi raccolti da $ per lavorare direttamente su ognuno di loro.

Inutile dire che $ accetta come primo argomento anche riferimenti diretti ad elementi del DOM:

$(document.getElementById('mioId'));  
$(window);

### Creare nuovi elementi

Proprio per rispettare la sinteticità del progetto la stessa funzione $ può essere utilizzata anche per altri scopi, che verranno interpretati dalla funzione in base agli argomenti passati. Così è possibile creare un nuovo elemento nel DOM semplicemente passandone il tag:

$("<div></div>");

o addirittura:

$("<div/>");

In questo modo possiamo anche creare elementi complessi in un’unica stringa di codice:

var blocco = $("<div><p>Esempio di elementi <strong>nidificati</strong></p></div>");

Comunque avremo la possibilità di applicare tutti i metodi di jQuery ai nuovi elementi:

blocco.addClass("nuovaClasse"); //aggiungo una classe al div

Infine, per inserire questi elementi all’interno della pagina useremo, per esempio, il metodo appendTo:

blocco.appendTo(document.body); //inserisci nel body  
blocco.appendTo("#contenitore"); //inserisci dentro un altro elemento

Ecco un [**esempio**](http://www.html.it/guide/esempi/jquery/esempi/core-1.html) pratico.

### Lanciare funzioni al caricamento del DOM

Una caratteristica particolare di $ è quella che permette di passare come argomento una funzione. In questo caso le istruzioni passate verranno lanciate “on DOM ready“, cioè quando l’albero degli elementi HTML è stato caricato:

$(function () {

alert("DOM caricato!");

});

Questo script differisce da:

window.onload = function () {

alert("Caricato!");

};

in quanto nel secondo la funzione viene lanciata quando tutto il documento è stato caricato, compresi i fogli di stile e le immagini. L’evento DOM ready, invece, permette di associare funzioni agli elementi del documento con la sicurezza che essi verranno eseguiti anche se l’utente li richiama prima del caricamento completo della pagina ([**esempio**](http://www.html.it/guide/esempi/jquery/esempi/core-2-domready.html)).