jQuery: come installare e creare un plugin partendo da zero?

Uno dei punti di forza di molti **frameworks javascript** (nella fattispecie **jQuery**) sta nella possibilità, e spesso nella facilità, di **creare estensioni** che possono migliorare o semplificare ulteriormente il nostro lavoro.

La potenzialità dei plugin sta, a loro volta, nella facilità di **utilizzo e personalizzazione** tramite parametri (le opzioni del plugin) i quali possono permetterci di **impostare** un **colore di sfondo** piuttosto che il **tempo di durata** di un'animazione.

Che cos'è un plugin?

Nel nostro campo un plugin non è altro che **un'estensione** (non autonoma) di un programma, che ha come compito quello di **ampliarne le funzioni.**

Più nello specifico un plugin **è una**, o un **insieme**, **di funzioni** che hanno lo scopo di utilizzare il framework per **compiti pre-stabiliti** e quindi aiutare noi sviluppatori quando si tratta di validare i nostri formulari, creare delle slideshow, inserire nei nostri siti delle gallerie fotografiche etc etc.

Come avviene l'installazione di un plugin?

In jQuery **l'installazione di un plugin** è (solitamente) abbastanza veloce, poichè tutto quello che c'è da fare è **includere i file necessari** al suo funzionamento nelle pagine del nostro sito ed in seguito **richiamare la funzione del plugin** (come vedremo più avanti).
Ovviamente, il tempo e la difficoltà di installazione varia da plugin a plugin.

A rendere il lavoro molto più facile è la **documentazione** e spesso anche **le demo** che possiamo trovare nella pagina del plugin che vogliamo installare oppure nel repository (wikipedia: repository) ufficiale di jQuery che trovate a questo indirizzo.

Installiamo insieme un plugin per jQuery

Vediamo insieme come avviene l'installazione di un plugin, **dalla ricerca alla scelta** e **dall'installazione al funzionamento**.

In questo esempio installeremo un plugin per visualizzare le immagini del nostro sito con il famoso effetto "lightbox".

La prima cosa da fare è **cercare tra i plugin di jQuery** ciò di cui abbiamo bisogno. Per la ricerca abbiamo a disposizione il repository di jQuery oppure un qualsiasi motore di ricerca.

Inserendo in Google le parole chiave "jquery lightbox" verrano visualizzati più di 1.000.000 di risultati; quello che ci resta da fare è scegliere quello che ci sembra il più adatto al nostro progetto tenendo in considerazione alcuni fattori:

1. **Il plugin deve essere crossbrowser**
   É essenziale che il plugin sia crossbrowser, questo significa che l'estensione che utilizzeremo deve essere compatibile con tutti i principali browsers (wikipedia: crossbrowser).
2. **La documentazione deve essere chiara e comprensibile**
   Non è mai semplice "prendere in mano" il lavoro fatto da qualcun'altro, per questo motivo la documentazione del plugin deve essere il più chiara possibile, in modo da permetterci l'utilizzo del plugin senza particolari sforzi.
   Inoltre una documentazione ben fatta ci permette di conoscere a pieno le potenzialità del plugin, oppure i suoi limiti.
3. **Vedere il plugin in azione può semplificare la nostra ricerca**
   La maggior parte dei plugin possiede una pagina di demo dove è possibile vederlo in funzione e quindi valutare se l'effetto è ciò di cui avevamo bisogno.

Per questo esempio ho scelto "jQuery lightBox plugin di Leandro Vieira" perchè nel sito del plugin la documentazione è buona, il plugin sembra facile da installare e la demo mi è piaciuta.

Dopo aver scaricato il file zip contenente i file necessari per il funzionamento e la demo per vederne il risultato finale, mi dirigo subito nella sezione **How to use** nel sito dell'autore.

Osserviamo con **attenzione i seguenti passaggi**, poichè saranno quasi sempre una **consuetudine al momento di installare un plugin**:

1. **Inclusione dei file javascript**
   Come ogni plugin di jQuery, anche questo ha bisogno della libreria (il core di jQuery) per funzionare, non dimentichiamoci quindi di caricare il file del plugin dopo aver caricato la libreria.

   Inseriamo il seguente codice all'interno del tag del nostro sito:

   Chi ha già familiarità con i selettori di jQuery dovrebbe avere già capito quali sono le differenze tra i vari esempi, ma un ripassino non fa mai male:

   Esempio 1: la funzione viene associata ad ogni ancora presente nel nostro sito il quale attributo rel contenga la parola "*lightbox*"
   Esempio 2: la funzione viene associata ad ogni ancora contenuta nel div avente id "*gallery*"
   Esempio 3: questa volta la funzione viene associata ad ogni ancora che abbia la classe lightbox
   Esempio 4: in questo caso, la funzione viene associata ad ogni ancora presente nel nostro markup html

Ora che abbiamo seguito questi quattro semplici passi dovrebbe essere tutto pronto per il funzionamento del plugin.

Ecco il mio risultato: link

Dopo aver visto come **installare un plugin** possiamo procedere con la parte (forse) più interessante, ovvero come realizzarne uno partendo da zero.

Realizzare un plugin jQuery: iniziare dalla documentazione è sempre un buon inizio!

Quando si inizia un nuovo progetto, come la **creazione di un plugin per jQuery**, è buona prassi partire dalla documentazione ufficiale, sempre che quest'ultima sia di qualità e dettagliata.

In questo caso abbiamo a disposizione la documentazione ufficiale di jQuery (sulla creazione di plugins).

Come iniziare, gli approcci disponibili

jQuery rende disponibili due tipologie di **approcci per la creazione di un plugin**, vediamoli di seguito:

Primo approccio: La Funzione

Il codice appena visto si occupa di creare un **nuovo metodo** che viene aggiunto all'oggetto jQuery; il metodo si chiama *log* ed accetta un *parametro*.

Per richiamare il metodo appena visto utilizziamo il seguente codice:

Abbiamo visto che tramite l'utilizzo del primo approccio possiamo creare una funzione che è possibile richiamare in seguito; ora vediamo qualche caso nel quale possiamo utilizzare questo approccio:

In questo esempio la funzione che abbiamo creato ci permette di **fare il debug** delle nostre variabili (per esempio) in modo semplice ed elegante: se il nostro browser possiede la console, la funzione la utilizzerà per mostrare il contenuto del parametro; in caso contrario il contenuto verrà visualizzato tramite un alert.

Il funzionamento del codice appena descritto si può vedere in questo file di esempio: link

Vediamo ora un altro piccolo esempio sempre utilizzando il primo approccio.
Questa volta il seguente codice è tratto da un esempio reale; vi spiego di cosa si tratta.

Chi ha già creato qualche volta un tema per wordpress magari conosce la funzione "*is_page()*", la quale restituisce il valore "*true*" o "*false*" nel caso ci si trovi, o meno, nella pagina passata come parametro alla funzione "*(is_page('contatti'))*".

Essendo questa funzione molto comoda, ho deciso di creare un piccolo esempio della stessa in javascript utilizzando l'approccio che abbiamo visto in precedenza:

Il funzionamento di questo codice è molto semplice:

1. assegnamo alla variabile denominata "*url*" l'url attuale

2. tramite il metodo split(), creiamo un array con le vari parti di url, divise dal simbolo '/'

3. tramite il metodo reverse() invertiamo gli indici dell'array
   var page = page.reverse();
4. controlliamo se il parametro richiesto coincide con la pagina attuale

Anche in questo caso potete vedere in funzione l'esempio: link

Secondo approccio: Il Metodo

Il codice appena visto ci permette di chiamare il plugin e passargli anche il parametro **shaker = true**.

A differenza del primo, il secondo approccio è sicuramente il più utilizzato nella creazione dei plugins poichè ci permette di avere **totale controllo sull'elemento selezionato**, in questo caso tutti i formulari che abbiano la **classe validation**.

Alias personalizzato all'interno delle funzioni

Tutti noi che usiamo jQuery siamo abituati ad utilizzare la variabile **$**, ma se, ci capitasse in qualche progetto di dover utilizzare due framework differenti ?
In quel caso è molto probabile che l'utilizzo di **$** vada in conflitto con l'altro framework e quindi dovremmo utilizzare il metodo noConflict() per specificare un alias e sostituire nei nostri script il simbolo **$** con il nuovo alias.

Vediamo di seguito il modo per evitare questo problema.

Quanto appena visto ci permette di inserire tutto il codice del plugin all'interno di una function ed eseguirla automaticamente; questo avviene grazie alla presenza di *()*; .

Ora abbiamo utilizzato lo stesso codice, solo che questa volta è stato passato 'jQuery' come parametro della funzione.
Questo ci permette di utilizzare qualsiasi variabile di JavaScript come alias, in questo caso ancora **$**.

Diamo un nome al nostro plugin!

Tramite l'utilizzo di *$.fn.validation* stiamo creando una nuova funzione che per l'appunto si chiamerà **validation**.

Opzioni? Si, grazie!

A mio parere, la possibilità di personalizzare un plugin è essenziale per la riuscita di un buon lavoro, poichè questo permette anche agli utenti meno esperti di adattare il plugin alle loro

esigenze.
Vediamo di seguito come passare e configurare i parametri del plugin.

In questo esempio viene **creato un oggetto** (config) contenente i parametri di default per il funzionamento del plugin, inoltre, se il parametro options è definito, le opzioni di default **vengono "estese" utilizzando la funzione $.extend** di jQuery.

Per utilizzare le opzioni all'interno del plugin ci basterà usare il seguente codice:

Per personalizzare un parametro (o più) del nostro plugin sarà sufficiente dichiararlo al momento di eseguire la funzione del plugin:

Arrivati a questo punto abbiamo:

1. Usato un alias per ovviare il problema di conflitto
2. Creato una nuova funzione che contiene il codice del nostro plugin
3. Preparato i parametri di default e previsto la possibilità di personalizzarli.

Diamo un compito da eseguire al nostro plugin

Prima di procedere con la spiegazione è essenziale capire che il nostro plugin viene associato al selettore di jQuery (*$('a.test').validate();*), quindi gli elementi che possono "essere trovati", potrebbero rivelarsi uno o più, è per questo motivo che dobbiamo utilizzare un **ciclo For Each** in modo che il nostro plugin lavori su ogni elemento trovato.

Nel nostro caso utilizzeremo la **funzione each(); di jQuery** per ciclare **"this"**, che contiene, appunto, tutti gli elementi trovati da jQuery.

Permettiamo di accodare altre funzioni

Perchè al nostro plugin non manchi nulla dobbiamo permettere agli utenti di accodare altre funzioni di jQuery anche dopo aver chiamato il nostro plugin; per fare questo abbiamo bisogno di aggiungere **return this** prima della fine del plugin.

Questo permette di accodare altre funzioni, come in questo esempio:

Dare un nome al file

Solitamente, i plugin sviluppati per jQuery utilizzano la seguente formula per il nome:

nomeplugin.jquery.js

Quindi nel nostro caso il plugin si chiamerà:

validation.jquery.js

Un piccolo esempio per vedere il plugin in azione

Il compito che esegue questo plugin è molto semplice:
Inanzitutto, inseriamo il valore di **$(this)** nella variabile **$this**: facciamo questo solo per comodità, poichè ci farà risparmiare tempo ogni volta che avremmo bisogno di usare $(this).

Il secondo passo è controllare il valore del parametro "output":
se questo è uguale a "alert", allora mostriamo il contenuto dell'attuale elemento ciclato tramite un alert; viceversa se il valore è console, il contenuto lo mostriamo all'interno della console.

Demo

Potete vedere il plugin in funzione applicato ad ancore presenti nel nostro markup html: link

Conclusioni

In questo articolo abbiamo visto che ci sono due approcci per creare un plugin per jQuery:
La funzione ed il Metodo.

Abbiamo inoltre visto le principali differenze tra i due approcci:
Con il **primo** creiamo una **semplice funzione** che può essere chiamata in qualsiasi momento, mentre il **secondo** approccio ci dà un **totale controllo sull'elemento ciclato**.

La facilità di estensione di questo framework è uno dei suoi punti di forza, infatti grazie a poche righe di codice siamo già riusciti a **creare un plugin per jQuery**.

Ora che hai imparato come si creano le estensioni di jQuery, quale sarà **il tuo primo plugin** ?