Far fronte agli eventi… in JavaScript - Parte 2

Capita sempre più spesso di avere a che fare con eventi che si scatenano decine di volte al secondo, come *resize*, *scroll*, *mousemove* e *touchmove*. I casi più frequenti sono quelli di *resize* e *scroll* dato che ci troviamo in un'epoca in cui si usa molto il cosiddetto effetto *parallax* e in cui un layout responsive è fondamentale, ma il supporto alle ultime specifiche CSS da parte dei browser non è ancora completo.

Altre volte, invece, abbiamo bisogno di aggiornare una chat, elaborare il testo di una *textarea* man mano che l'utente digita, gestire il contenuto di in un *canvas*. Come potrete immaginare, scatenare eventi che interagiscono con il DOM così frequentemente comporta problemi di performance: come farvi fronte?

Debouncing

Si parla di **Debouncing** quando si fa in modo che una funzione venga eseguita soltanto se è trascorso un certo lasso di tempo dall'ultima volta in cui sarebbe stata **chiamata** (quindi, verrà eseguita soltanto una volta, l'ultima).

Facciamo l'esempio di un evento che, quando si sta ridimensionando la finestra del browser, cambia l'altezza di un div con id box2 per mantenerla uguale a quella di un div con id box1. Per praticità gestiremo il tutto con jQuery.

Il DOM viene interrogato e manipolato ad ogni variazione di dimensioni della finestra, scatenando decine di eventi mentre stiamo trascinando il bordo del browser, con la probabile percezione di "scatti" da parte dell'utente.

Non sarebbe meglio, quindi, eseguire un solo evento al termine del ridimensionamento del browser? Decisamente sì. In pratica si può usare un setTimeout per rimandare di "n" millisecondi l'esecuzione della funzione associata al *resize*. Inoltre, si "resetta" il timer se l'evento si manifesta nuovamente entro l'intervallo di "n" millisecondi, ricominciando il conteggio da capo.

Throttling

Si parla di **Throttling** quando si fa in modo che una funzione venga eseguita soltanto se è trascorso un certo lasso di tempo dall'ultima volta in cui è stata **eseguita** (quindi verrà eseguita

“ogni tot”).

Un esempio può essere il riposizionamento di un *div* sull’asse delle ordinate (y) in base allo scroll di una pagina. Facciamo un esempio, dando per assunto che:

- per motivi di studio, procederemo “alla vecchia maniera”, posizionando l’elemento in javascript ed usando position:absolute invece di position:fixed
- la pagina sarà tanto lunga da potersi scrollare ad esempio per un paio di secondi
- vogliamo dare un feedback visivo del fatto che il box resterà sempre alla stessa distanza dall’angolo in alto a sinistra del viewport riposizionandolo ad ogni scroll della pagina

Anche qui, per praticità useremo jQuery.

Uno script non ottimizzato potrebbe essere questo:

Invece, potremmo ottimizzarlo applicando il concetto di throttling usando un setTimeout e una variabile timeoutHandler:

Semplificare l’implementazione

A seconda dei casi, l’implementazione può variare. Ad esempio, potremmo voler eseguire la funzione per la prima volta dopo il timeout oppure immediatamente, senza aspettare (vale anche la pena chiedersi se queste differenze siano poi così importanti). Oppure, potremmo voler usare requestAnimationFrame anziché setTimeout. Molti sviluppatori hanno già affrontato queste situazioni, creando degli script di qualità che hanno messo a disposizione della comunità. Vi consiglio, ad esempio, il plugin di Ben Alman (su cui, tra l’altro, si basano le implementazioni delle stesse funzionalità di *Underscore.js*) che può funzionare anche senza jQuery e che rende le cose molto più semplici:

Coming soon

Nella terza parte di questo articolo parleremo di come lo sviluppo di siti web mobile necessiti a volte dell’implementazione di eventi particolari. È facile, infatti, avere a che fare con *multitouch* e *gesture*: vedremo quindi come far fronte anche a questi eventi!

Leggi anche:

- Far fronte agli eventi... in JavaScript - parte 1
- Far fronte agli eventi... in JavaScript - parte 2