Far fronte agli eventi… in JavaScript - Parte 1

Da diversi anni jQuery ha semplificato notevolmente la gestione degli eventi del DOM. Tuttavia, ancor oggi ogni sviluppatore web prima o poi si imbatte in alcune situazioni tipiche nelle quali bene o male riesce ad arrangiarsi. Questo articolo vuole spingersi un po' oltre, approfondendo le tematiche e dando nuovi strumenti per affrontare... gli eventi!

Il codice riportato in questo articolo è compatibile con tutti i principali browser tranne quelli molto vecchi, come Internet Explorer 8 e precedenti, per i quali esistono semplici fix facilmente reperibili.

Bubbling e Capturing

Ripassiamo le basi e andiamo a guardare la specifica di *addEventListener()*, il metodo raccomandato per sfruttare gli eventi.

Se dovessimo far comparire in un alert la parola 'grazie' al click di un elemento con id "cliccami" potremmo scrivere:

Molti sviluppatori tendono a non indicare il parametro *useCapture* dato che è diventato opzionale poco dopo la sua introduzione perché molto spesso il codice funziona con il valore di default (*false*). E infatti, anche aggiungendo *true* come terzo parametro di *addEventListener()* nell'esempio precedente il risultato non cambia.

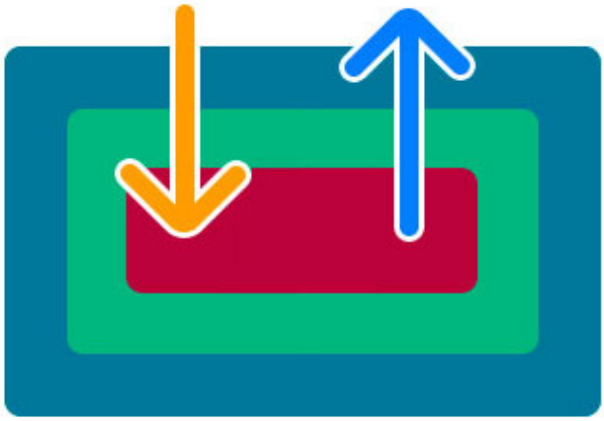
Ma, allora, cosa rappresenta il parametro *useCapture*, e in quali casi torna utile usarlo?

Partiamo da un piccolo esempio.

Cliccando sul link, nella console verrà stampato *tre, due, uno*. Se aggiungessimo *true* come terzo parametro di *addEventListener()* otterremmo invece *uno, due, tre*. Possiamo quindi capire che *useCapture* determina l'ordine in cui vengono scatenati gli eventi. Ma come, esattamente? Ecco uno schema esplicativo.

Come funziona useCapture in addEventListener?

Prima fase: Capturing



Seconda fase: Bubbling

```
<ul id="uno">
    <li id="due">
        <a id="tre" href="#">Example</a>
    </li>
</ul>
```

Ogni volta che il browser fornisce un input adeguato, l'interrogazione degli elementi inizia. Per prima viene eseguita la fase di **Capturing**, per seconda quella di **Bubbling**. La fase di Capturing è anche chiamata di Trickling, che rende più facile ricordare il concetto grazie alla mnemotecnica "trickle down, bubble up" (gocciola in giù, bolle in su).

Possiamo quindi riassumere l'utilizzo del parametro *useCapture* così:

- *false (default)* = il listener viene agganciato alla fase 2, quella di Bubbling, mentre cioè gli elementi vengono interrogati dal più al meno annidato.
- *true* = l listener viene agganciato alla fase 1, quella di Capturing, mentre cioè gli elementi vengono interrogati dal meno al più annidato.

È possibile interrompere il propagarsi degli eventi usando due metodi:

- *event.stopImmediatePropagation()* = blocca gli eventi che si manifesterebbero successivamente a quello corrente
- *event.stopPropagation()* = blocca gli eventi associati agli elementi che verrebbero interrogati successivamente a quello corrente

Nell'esempio precedente, aggiungendo uno dei due metodi subito dopo il *console.log(this.id)* verrebbe stampato soltanto *tre*.

A cosa serve sapere tutto questo? Torna decisamente utile quando si ha a che fare con mouseover, drag'n'drop, eventi che non hanno la fase di bubbling (focus, blur, museenter, mouseleave, ecc) e altri pochi casi molto particolari. jQuery per semplicità scatena tutti gli eventi in fase di bubbling, quindi il suo utilizzo in questi ambiti è da valutare.

Event delegation

Diciamo di avere una pagina con una lista di un centinaio di elementi che cambiano a seconda delle azioni dell'utente. Al click su questi elementi deve succedere qualcosa. Il primo modo di procedere che viene in mente è quello di aggiungere un listener ad ogni elemento, creando così ben cento listener, senza contare che questi andrebbero rimossi e aggiunti ogni volta che le voci cambiano. La tecnica di programmazione di cui sto per parlavi permette invece di aggiungerne soltanto uno, e una volta per tutte.

Consiste nell'aggiungere un listener al contenitore: JavaScript è in grado di dirci su quale discendente abbiamo cliccato, perciò, possiamo discriminare le azioni a seconda dell'elemento che abbiamo cliccato, detto "target" dell'evento. L'esempio seguente loggherà le scritte *tre*, *cinque*, *sette* quando si cliccherà sull'elemento corrispondente.

È possibile farlo anche in jQuery ovviamente, in questo modo:

È interessante notare che usando jQuery, a *this* viene associato l'*event.target*.

Coming soon

Il prossimo evento a cui dovrete prestare attenzione è… la pubblicazione della seconda parte di questo articolo! Si parlerà di performance, capendo come gestire quegli eventi che si scatenano decine di volte al secondo, come *resize*, *scoll*, *mousemove* e *touchmove*.

Leggi anche:

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