WordPress: breve guida all'uso dei CSS e di JavaScript

Come usare i CSS e JavaScript in WordPress.

Inclusione del codice JavaScript e CSS

WordPress opera una distinzione tra il codice lato client inserito nel backend e quello inserito nel frontend, ossia nella sezione amministrativa e nel sito.

Ci sono due action distinte:

1. **admin_enqueue_scripts**
2. **wp_enqueue_scripts**

La prima action è per il backend, la seconda per il frontend. Anche se il nome può trarre in inganno, queste due action servono per includere i file JavaScript e CSS.

Esempio:

```
function add_js() {
    wp_register_script( ‘my’, plugins_url() .
‘/plugin/js/my.js’ );
    wp_enqueue_script( ‘my’ );
}
add_action( ‘wp_enqueue_scripts’, ‘add_js’ );
```

Le funzioni da usare:

1. wp_register_script()

2. wp_enqueue_script()
3. wp_register_style()
4. wp_enqueue_style()
5. wp_script_is()
6. wp_style_is()

Per gli autori di plugin, anche se può sembrare strano, le funzioni più importanti per scrivere plugin flessibili sono le ultime due: servono infatti a verificare che uno script o un file CSS non sia già stato incluso.

Immaginiamo il caso in cui un tema usi jQuery, scenario molto comune. Il seguente codice ci eviterà di distruggere il frontend:

```
if( !wp_script_is( ‘jquery’ ) ) {
    wp_enqueue_script( ‘jquery’ );
}
```

WordPress dispone di molti script preinstallati, tra cui jQuery. In questo caso, dato che il nostro codice usa jQuery, dobbiamo prima verificare che non sia già presente prima di includerlo.

Se non avessimo effettuato questa verifica avremmo avuto due copie di jQuery sullo stesso sito, il che avrebbe generato errori fatali a cascata. Lo stesso principio si applica per i file CSS.

Struttura del codice JavaScript

WordPress, specialmente con jQuery, spinge gli autori ad usare un namespace per il proprio codice JavaScript. Avrete sicuramente visto questo modo di usare jQuery in WordPress:

```
(function( $ ) {
    // codice
})( jQuery );
```

Si tratta di una funzione self-executing che assegna alla variabile $ l'oggetto jQuery. In questo modo possiamo usare jQuery direttamente con il suo alias.

In JavaScript quando usiamo questo pattern creiamo una sandbox in cui il nostro codice può si operare sulle pagine ma resta isolato dal resto del codice JavaScript presente nel sito.

Questo è fondamentale, dato che noi non possiamo sapere se i nomi delle nostre variabili, delle funzioni e degli oggetti siano usati già da qualche altro script già usato da un plugin o dal tema.

A differenza di PHP, JavaScript non interrompe l'esecuzione del codice se una variabile ad esempio viene ridefinita. Molto semplicemente, la variabile creata dopo la prima ne sovrascrive il valore. La stessa cosa, purtroppo, vale anche per le funzioni e per gli oggetti.

Oltre alla sandbox che abbiamo già visto, possiamo ulteriormente rafforzare il nostro codice usando un prefisso da usare con funzioni e oggetti. Un'altra cosa fondamentale è quella di evitare ad ogni costo le variabili globali ed affidarci invece al contesto (scope) creato dal nostro codice.

Struttura del codice CSS

Il principio della creazione di un namespace per il nostro codice si applica anche ai CSS con una fondamentale differenza: nei CSS i prefissi sono l'unico sistema.

Possiamo aggiungere un prefisso ai selettori di classe e di ID in questo modo:

```
#myns-container { … }
.myns-item { … }
```

mioplugin è il nostro prefisso che non solo ci mette al riparo da possibili conflitti, ma permette anche agli autori ed agli utenti di modificare i nostri stili con maggiore facilità.

Una volta creato un namespace possiamo sfruttare il contesto creato dai selettori per creare regole più specifiche:

```
#myns-container .myns-item a { … }
```

Sfruttando la cascata possiamo indirizzare le nostre regole con maggiore precisione. Se vi state chiedendo come gli utenti possano sfruttare il nostro namespace creato dai prefissi, ecco un esempio:

```
[class^=“myns”] * {
    font-family: Arial, sans-serif !important;
}
```

In questo caso gli utenti possono effettuare un reset globale sui nostri elementi senza dover conoscere necessariamente il nome dell’elemento. Un notevole vantaggio.