WordPress e la gestione degli errori PHP

La gestione degli errori in WordPress ricade in parte sotto la gestione comune degli errori in PHP ed in parte sotto la gestione degli errori specifica di WordPress. In questo articolo vedremo come gestire gli errori in WordPress.

Errori di sintassi

Gli errori più comuni sono quelli riguardante la sintassi di PHP. Il modo migliore per prevenire questo tipo di errori è quello di dotarsi di un editor che esegua un check sulla sintassi prima di salvare il file sul server.

Da evitare l'uso dell'editor interno di WordPress, perché questo editor non solo non dispone delle funzioni più elementari ma non esegue nessun tipo di controllo sulla sintassi PHP.

Gli errori di sintassi generano un **Fatal Error** in PHP che blocca completamente l'esecuzione del resto del codice. Qualora il vostro server sia configurato per non mostrare i messaggi di errore di PHP ma di registrarli in un file di log, l'errore restituito dal server sarà il 500.

Errori derivanti da classi e funzioni non definite

L'interprete PHP blocca l'esecuzione del codice quando si cerca di usare una funzione che non è stata definita. Il caso più tipico in WordPress è quando usiamo una funzione di un plugin nel nostro tema e quindi disabilitiamo il plugin.

Disabilitando il plugin la funzione non esiste più nel flusso di WordPress, quindi è undefined. Prevenire questo tipo di errore è possibile tramite la funzione PHP `function_exists()`:

```
if( function_exists( 'my_plugin_func' ) ) {
    my_plugin_func();
}
```

In modo analogo possiamo verificare che una classe sia stata definita utilizzando la funzione PHP `class_exists()`:

```
if( class_exists( 'MyClass' ) ) {
    $myClass = new MyClass();
}
```

L'errore più comune con le classi si manifesta quando cerchiamo di invocare un metodo di una classe. In questo caso l'interprete PHP ci segnalerà che stiamo cercando di usare un metodo su un non-oggetto (non-object).

Errori derivanti da input imprevisto

Supponiamo di aver creato una funzione che genera un Loop tramite la classe `WP_Query` a partire da un array associativo di parametri passato alla funzione. Cosa succederebbe se l'input della funzione non è un array?

```
function make_loop( $args ) {
    $loop = new WP_Query( $args );
}
```

```
        //...  
    }
```

Ci sono due soluzioni a questo problema:

1. Usare un valore predefinito
2. Effettuare una verifica sui dati

Ecco la prima soluzione:

```
function make_loop( $args = array() ) {  
    $loop = new WP_Query( $args );  
    //...  
}
```

Questa soluzione può essere migliorata ulteriormente:

```
function make_loop( $args = array() ) {  
    $defaults = array(  
        'posts_per_page' => 3  
    );  
    $args = wp_parse_args( $args, $defaults );  
  
    $loop = new WP_Query( $args );  
    //...  
}
```

In questo modo la nostra funzione avrà degli argomenti predefiniti che le permetteranno comunque di restituire un output.

La seconda soluzione consiste nell'utilizzare le funzioni PHP di verifica del tipo di dati:

```
function make_loop( $args ) {
    $loop_args = array();
    if( is_array( $args ) && count( $args ) > 0 ) {
        // è un array e non è vuoto
        $loop_args = $args;
    } else {
        $loop_args = array(
            'posts_per_page' => 3
        );
    }
    $loop = new WP_Query( $loop_args );
    //...
}
```

Questo approccio non vale solo per le funzioni, ma più in generale per tutte quelle routine che richiedono l'uso di un input definito da codice esterno.

Errori derivanti dai valori di ritorno delle funzioni di WordPress

Consideriamo il seguente codice:

```
global $post;
$id = $post->ID;

$terms = get_the_terms( $id, 'nometassonomia' );
```

```
foreach( $terms as $term ) {
    echo $term->name . "\n";
}
```

L'errore di fondo sta nel fatto che non abbiamo considerato i valori di ritorno della funzione `get_the_terms()` nel caso in cui non esistano termini associati al post o la tassonomia specificata non esista.

Quindi:

```
global $post;
$id = $post->ID;

$terms = get_the_terms( $id, 'nometassonomia' );

if ( $terms && ! is_wp_error( $terms ) ) {

    foreach( $terms as $term ) {
        echo $term->name . "\n";
    }
}
```

La sezione **Returns** (valori di ritorno) della documentazione di WordPress per questa funzione afferma infatti che:

Array of term objects on success; false if the post contains no terms from the given taxonomy; false if the post doesn't exist; and a `Class_Reference/WP_Error` object if an invalid taxonomy is entered.

La funzione `is_wp_error()` verifica che il valore restituito sia o meno un'istanza della classe core `WP_Error`. Questa classe può essere usata anche per gestire i nostri errori:

```
function my_func( $param ) {
    if( !$param ) {
        return new WP_Error( 'codice-errore',
'Messaggio di errore' );
    }
}

$return = my_func();
if( is_wp_error( $return ) ) {
    echo $return->get_error_message(); // 'Messaggio
di errore'
}
```

Il costruttore della classe accetta tre parametri:

1. una stringa rappresentante il codice dell'errore
2. il messaggio di errore associato al codice
3. i dati associati all'errore sotto forma di array

Potete trovare maggiori informazioni su questa classe nella documentazione ufficiale.

Conclusione

Gestire gli errori in WordPress è un'operazione che richiede la conoscenza di PHP e del funzionamento interno delle classi e funzioni di WordPress. Avendo presente questi due aspetti il compito è enormemente semplificato.