PHP E WORDPRESS: COSA C'È DA SAPERE

Gabriele Romanato

http://gabrieleromanato.com
http://wordpressista.com

PHP?

WordPress è scritto in PHP, ma a giudicare dal numero di richieste di supporto molti utenti e sviluppatori spesso se ne dimenticano.
Scopo di questo intervento è quello di delineare con chiarezza quali sono le caratteristiche di PHP
che è necessario conoscere per poter sviluppare temi e plugin di WordPress senza troppi problemi.

-

SI, È SOLO PHP

- Tutto quello che sapete su PHP si applica anche a WordPress.
- Tutto quello che sapete su WordPress è basato su PHP.
- Tranne il codice lato client (CSS, HTML5, JavaScript).
- Chi conosce PHP ha già un'ottima conoscenza di WordPress.
- Quello che rimane da fare è studiare la documentazione.
- https://codex.wordpress.org/
- STUDIARE LA DOCUMENTAZIONE È FONDAMENTALE
-

COMINCIAMO?

- NON ABBIATE PAURA DI SBAGLIARE!
- Come con qualsiasi altro linguaggio, in PHP è fondamentale non avere paura di sbagliare.
- Per imparare dovete creare un vostro spazio in locale dove effettuare i vostri test.
- http://www.ampps.com/
- AMPPS ad esempio vi permette di avere Apache, MySQL, PHP senza troppi problemi.
- Test, test e ancora test!
-

COSA FA PHP?

- Gran parte del lavoro di PHP avviene dietro le quinte della pagina.
- Il risultato del lavoro di PHP lo apprezziamo quando questo genera il codice HTML o il contenuto finale.
- Avrete visto il seguente codice n-mila volte in WordPress:

- `<?php the_content(); ?>`

- PHP in questo caso si collega al database, seleziona il campo `post_content` del post corrente e manda in output tutto
- quello che è stato inserito dall'utente nell'editor del post di WordPress.
- E quello che avremo alla fine sulla pagina è solo codice HTML.

TUTTO QUI?

- CERTO CHE NO!
- Oltre ad interagire con il database, con PHP potete operare sui file, generare e manipolare immagini,
- dialogare con un server remoto, usare i protocolli FTP, IMAP, SMTP, POP3 ecc., ecc.
- L’unico limite è dato dalle impostazioni del vostro server (lo vedremo più avanti) e dalle specifiche
- del progetto che volete realizzare.
- Insomma ce n’è per tutti i gusti!

REQUISITI RICHIESTI

- WordPress richiede una versione di PHP uguale o superiore alla 5.2 per poter funzionare.
- Dovreste sempre verificare la versione di PHP prima di iniziare un progetto in WordPress.
- Solitamente gli hosting provider forniscono queste informazioni
- ma qualora non fosse possibile dovete semplicemente creare un file .php
- con questo codice:
- `<?php phpinfo(); ?>`
- In questo modo aprendo il file nel browser avrete tutte le informazioni sull'installazione di PHP in uso.
- Non dimenticate di rimuovere il file dopo averlo visualizzato o salvato: è un enorme rischio lasciarlo sul server così com'è.

MODALITÀ DI DEBUG

- Prima di rilasciare un progetto in produzione
- dovreste sempre abilitare la modalità di debug di WordPress.
- Questo vi permetterà di visualizzare tutti gli errori e gli avvisi che WordPress solitamente nasconde.

- Nel file wp-config.php della vostra installazione modificate:

- `define( 'WP_DEBUG', false );`

- in

- `define( 'WP_DEBUG', true );`

- Ovviamente quando siete in produzione il valore va reimpostato su `false`.

IL CODICE PHP DOVE VA?

- REGOLA ASSOLUTA: MAI, MAI E POI MAI MODIFICARE I FILE DEL CORE DI WORDPRESS!!!
- Ossia tutti i file .php che si trovano nella root del vostro sito e nelle directory /wp-admin e /wp-includes.

- Avete invece a disposizione alternative più valide e sicure:

- I file template del vostro tema (header.php, index.php ecc.) e il file functions.php del vostro tema dove potete anche
- includere codice esterno al file stesso.

- Il file principale del vostro plugin dove potete anche includere codice esterno al file stesso.

- Usando queste alternative, WordPress può inserire il vostro codice nel suo flusso di esecuzione.

HO UN DUBBIO...

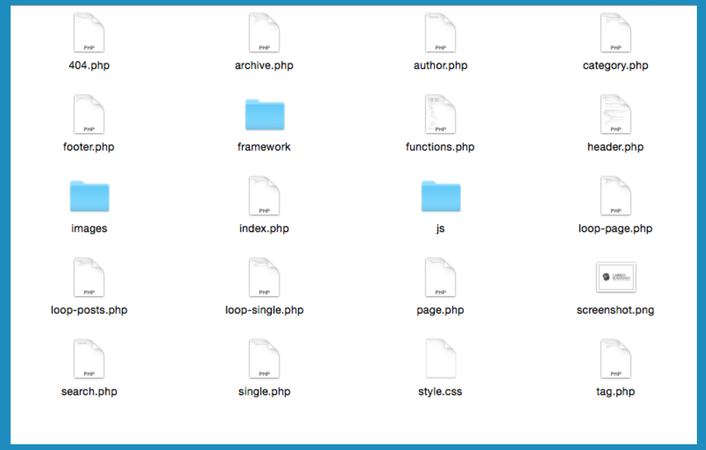
- Prima di cercare disperatamente una risposta su Google, spesso finendo per usare una soluzione
- poco adatta o addirittura dannosa, due sono i siti che dovete tenere sempre presenti:

- 1
- Documentazione ufficiale di PHP
- http://php.net/manual/en/index.php

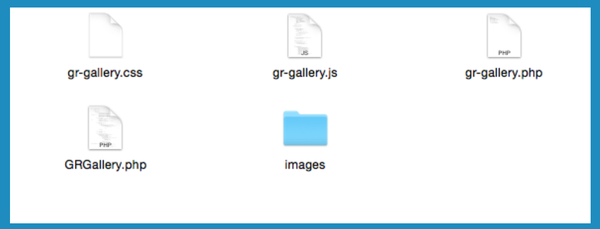
- 2
- Documentazione ufficiale di WordPress
- https://codex.wordpress.org/

- Scoprirete che a volte una soluzione macchinosa con PHP può essere risolta da una sola funzione di WordPress,
- o che una feature mancante in WordPress può essere sostituita da una semplice soluzione in PHP.

FILE E DIRECTORY



- **STRUTTURA DEI FILE E DELLE DIRECTORY DI UN TEMA**



- **STRUTTURA DEI FILE E DELLE DIRECTORY DI UN PLUGIN**

FILE E DIRECTORY 2

- **STRUTTURA DEI FILE E DELLE DIRECTORY DI UN TEMA**

- Un tema deve avere necessariamente un file index.php e un file style.css.
- I file .php che avete visto nella directory principale del tema sono i file template del tema stesso.
- functions.php è un file speciale che serve a far eseguire il nostro codice PHP nel flusso di WordPress.
- In questo caso functions.php richiama le classi PHP contenute nella directory **/framework**:

- `require_once( TEMPLATEPATH . '/framework/My_Class.php' );`

- TEMPLATEPATH è una costante di WordPress che memorizza il path assoluto alla directory del tema corrente.
- Non è necessario utilizzare classi: potete anche includere file .php con funzioni se avete scelto di non usare
- un approccio OOP (Object-Oriented Programming):

- `require_once( TEMPLATEPATH . '/framework/my_funcs.php' );`

- Se il vostro codice è complesso e strutturato è sempre bene non inserirlo tutto solo nel file functions.php.

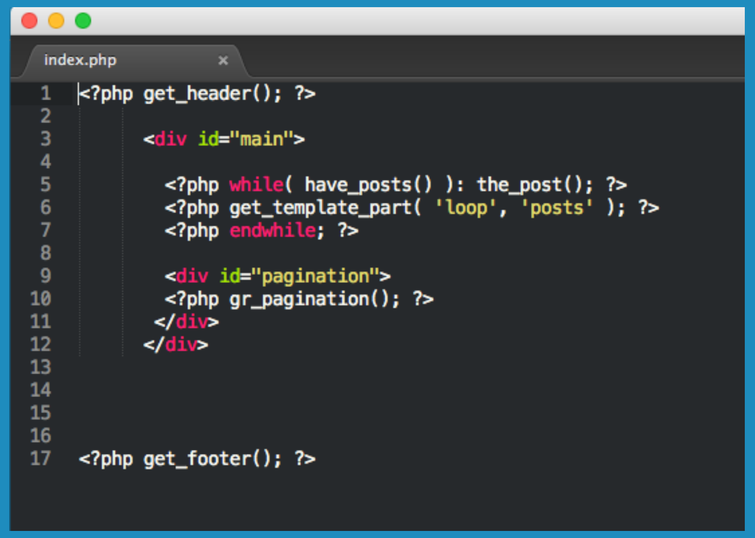
FILE E DIRECTORY 3

- **STRUTTURA DEI FILE E DELLE DIRECTORY DI UN PLUGIN**

- L’unico vincolo strutturale per un plugin è che il suo file principale .php deve avere lo stesso nome della sua directory.
- Quindi se la directory è **my-plugin** il file dovrà essere **my-plugin.php**

- Per il resto l’autore di un plugin può strutturare i suoi file e le directory come meglio crede
- anche se quello che abbiamo detto a proposito della struttura del codice PHP di un tema
- si applica anche ai plugin.

I FILE TEMPLATE DEI TEMI



SPIEGAZIONE

- Questi file sono composti sia da codice PHP che HTML semplice.

- Il codice PHP viene racchiuso tra `<?php` e `?>`
- mentre il codice HTML viene inserito direttamente.

- Il codice PHP in questo caso si divide nei costrutti fondamentali del linguaggio
- (come il ciclo `while()`)
- e in chiamate a varie funzioni, come `get_header()` e `gr_pagination()`.

- La prima è una funzione di WordPress, la seconda è una funzione definita dal tema in uso.
- La prima cosa da imparare è distinguere tra le funzioni core di WordPress e quelle definite dal tema.
- Gli autori di temi dovrebbero sempre usare un prefisso come `gr_` per facilitare il riconoscimento di questo tipo di funzioni.

- Quello che vedete all'interno delle parentesi tonde delle funzioni sono gli **argomenti delle funzioni**.
- Una funzione può avere zero o più argomenti.

-

IL LOOP

```
<?php
while( have_posts() ):
the_post();
endwhile;
?>
```

oppure

```
<?php
while( have_posts() ) {
the_post();
}
?>
```

- Il Loop di WordPress è un semplice ciclo PHP.
- Finché ci sono post da visualizzare vengono inizializzati i dati del post corrente tramite la funzione `the_post()`.
- Quando la funzione `have_posts()` restituisce `false` il ciclo si interrompe.

IL LOOP 2

- Il Loop così come lo abbiamo visto ovviamente non visualizza i contenuti dei post.
- Per farlo abbiamo bisogno di combinare il codice HTML con le funzioni di WordPress per i post:

```
<?php while( have_posts() ): the_post(); ?>
<h2 class="post-title"><?php the_title(); ?></h2>
<p class="post-excerpt"><?php the_excerpt(); ?></p>
<?php endwhile; ?>
```

- Come ci suggerisce il loro nome, le due funzioni di WordPress che abbiamo usato servono a visualizzare rispettivamente il titolo e il riassunto del post.

- Man mano che il Loop avanza queste funzioni prendono i dati del post corrente.
- Ovviamente esistono molte altre funzioni per i post.
- Quello che dovete verificare nella documentazione è se la funzione deve necessariamente essere usata solo nel Loop.
- Ad esempio queste due funzioni hanno effetto solo nel Loop.

COSTRUTTI CONDIZIONALI

COSA SUCCEDE SE NON CI SONO POST?

COSA SUCCEDE SE IL POST NON HA UN RIASSUNTO?

Possiamo combinare il costrutto condizionale PHP `if...else` con le funzioni condizionali di WordPress, ossia:

se(if) la condizione è vera
esegui questo codice
altrimenti (else)
esegui questo codice alternativo

```
<?php if( have_posts() ) {
while( have_posts() ) {
the_post();
}
} else {
?>
<p>Oops, nessun post! :-(</p>
<?php } ?>
```

```
<?php if( has_excerpt() ) { ?>
<p class="post-excerpt"><?php the_excerpt(); ?>
<?php } else { ?>
<p class="post-excerpt"><?php echo substr( get_the_excerpt(), 0, 150 ); ?>…</p>
<?php } ?>
```

VARIABILI

- Vogliamo aggiungere una classe CSS speciale a tutti i post pari del Loop, ossia 2, 4, 6 e così via.
- Per farlo dobbiamo creare un contatore che parte da zero e viene incrementato di 1 ad ogni step del Loop.

- Possiamo usare una variabile.
- Una variabile in PHP inizia con il simbolo $ seguito dal nome della variabile.

```
<?php $counter = 0;
while( have_posts() ):
the_post();
$counter++; // 1, 2, 3 ecc.
?>
<article class="post<?php if( $counter % 2 == 0 ): ?> special<?php endif; ?>">
</article>
<?php endwhile; ?>
```

- % è l'operatore modulo di PHP e restituisce il resto di una divisione.
- Quindi se il numero del contatore diviso 2 restituisce 0 allora è pari.

VARIABILI E CONTESTO

La variabile $counter vista prima è **globale** ossia è visibile e accessibile in tutto il contesto del nostro codice.
Ma se volessimo usare quella variabile nel contesto di una funzione o di una classe PHP
avremo una sorpresa:

PHP NON SI COMPORTA COME JAVASCRIPT!

```
$my_string = ‘Test’;

function my_encode_str() {
echo md5( $my_string ); // non funziona!!!
}
```

Dobbiamo invece usare la parola chiave `global`:

```
function my_encode_str() {
    global $my_string;
echo md5( $my_string ); // funziona!
}
```

Ecco spiegato perché nei temi e plugin di WordPress vediamo spesso questa parola chiave
unita alle variabili globali di WordPress come `$post.`

FUNZIONI: functions.php

Il nome di questo file è fuorviante: infatti questo file del tema può contenere non solo funzioni
ma tutta la logica e le interazioni del nostro codice PHP con WordPress.

COS'È UNA FUNZIONE?
È un modo per eseguire un blocco di codice con uno scopo preciso.

Una buona funzione non è una funzione che fa tutto:
una buona funzione è una funzione che fa bene un compito preciso.

Non ha senso a livello di gestione del codice
creare funzioni gigantesche:
un giorno potremmo riprendere in mano il nostro codice
e perdere tempo prezioso
cercando di capire cosa fa esattamente una funzione
con decine di righe di codice.

Per scrivere una buona funzione dovete per prima cosa chiedervi:
cosa voglio ottenere?
E poi:
qual'è il modo più efficace per ottenere quello che voglio?

Ma prima di tutto dovreste verificare che non esista già una funzione di PHP o di WordPress
che fa già quello che volete ottenere.

NON REINVENTATE LA RUOTA!

FUNZIONI: DETTAGLI

Una funzione in PHP viene definita mediante la parola chiave `function` in questo modo:

```
function nome_funzione( $argomento1, $argomento2, … ) {
// codice della funzione
}
```

E usata in questo modo:

```
nome_funzione( ‘Test’ );
```

Se invece una funzione non ha argomenti:

```
nome_funzione();
```

Una funzione può o non può restituire un valore di ritorno ottenuto dal codice al suo interno.
Quando restituisce un valore si usa la parola chiave `return`:

```
function my_post_exists( $id ) {
  $post_obj = get_post( $id );
return ( is_null( $post_obj ) );
}
```

In questo caso la funzione restituisce true o false a seconda se il post esiste o meno.
Possiamo usare la funzione come segue:

```
$post_id = 12;
if( my_post_exists( $post_id ) ) {
// il post esiste
} else {
// il post non esiste
}
```

OGNI VOLTA CHE USATE UNA FUNZIONE DI PHP O DI WORDPRESS
DOVRESTE SEMPRE VERIFICARE QUAL'È IL SUO VALORE DI RITORNO!

FUNZIONI: ARGOMENTI

IN PHP ESISTE IL PROBLEMA DELL'ORDINE DEGLI ARGOMENTI NELLE FUNZIONI.

Sia che si tratti di argomenti predefiniti o no,
sbagliare l'ordine degli argomenti è molto facile.
E gli effetti sono disastrosi, sia a livello di errori che di output delle funzioni.

WORDPRESS RISOLVE IL PROBLEMA BRILLANTEMENTE:

```
function my_func( $args ) {
  $default_args = array(
      'a' => 1,
     'b' => 'ok'
      );
$args = wp_parse_args( $args, $defaults );

    extract( $args, EXTR_SKIP );
  // Ora abbiamo $a e $b
        //…
        }
```

In pratica la funzione di WordPress `wp_parse_args()` fonde insieme gli argomenti predefiniti con quelli passati alla funzione sotto forma di array associativo.

Il risultato finale è del tutto analogo a quello ottenuto con il metodo `$.extend()` nei plugin jQuery.
In questo modo il problema dell'ordine degli argomenti è risolto.

WORDPRESS E OOP

- WordPress è un ottimo esempio di fusione tra OOP (Object-Oriented Programming) e programmazione procedurale basata sulle funzioni.

- In WordPress il funzionamento core del CMS è affidato alle classi (come WP_Query) mentre le interazioni utente sono delegate alle funzioni che utilizzano le classi del Core (come get_posts()).

- In questo modo gli autori di temi e di plugin possono interagire con WordPress senza dover necessariamente conoscere sempre i dettagli implementativi del funzionamento del Core.

- È comunque possibile usare direttamente le classi del Core (come appunto WP_Query) ma quando i task da effettuare sono molto semplici sono da preferire le funzioni.

- In pratica se WordPress offre già una funzione per un task andrebbe usata quella.

- Il bello è che la sottostante struttura delle classi è sempre accessibile qualora ce ne fosse bisogno.

CLASSI E OGGETTI

- In un vecchio manuale Java la differenza tra classi e oggetti veniva spiegata in questo modo:
- è la stessa differenza esistente tra gli stampi per dolci e i dolci stessi.

- Una classe fornisce a un oggetto le caratteristiche che quest'ultimo userà quando verrà
- utilizzato nel nostro codice.

- Quando usiamo questo costrutto:

- `$my = new My_Class();`

- $my è l'oggetto che utilizza le caratteristiche definite in My_Class.
- Tecnicamente si dice che $my è un'istanza della classe My_Class.

-

CLASSI

Una classe può essere definita in questo modo:

```
class My_Class {
public $proprieta;

public function __construct( $value ) {
$this->proprieta = $value;
}

public function metodo( $str ) {
return strtoupper( $this->proprieta ) . $str;
}
}
```

__construct() è il metodo magico che inizializza le proprietà di una classe.
Quando una proprietà è una funzione prende il nome di metodo.
public, private e protected sono i modificatori della visibilità di una proprietà di una classe.
public indica una visibilità universale.
protected indica che la proprietà è visibile nella classe stessa e nelle classi discendenti o genitori.
private indica invece che la proprietà è visibile solo nella classe che la definisce.
$this è un riferimento alla classe stessa.
-> è l'operatore con cui accediamo alle proprietà di una classe.

Esempio:

```
$my = new My_Class( 'Hello' );
echo $my->proprieta; // 'Hello'
echo $my->metodo( ' WORLD' ); // 'HELLO WORLD'
```

USO DELLE CLASSI

Per usare le classi in WordPress è consigliabile seguire lo stesso design del CMS:
alle classi verrà delegata la gestione del Core del nostro tema o plugin
mentre le funzioni useranno queste classi per poter interagire con la parte frontend
o backend di WordPress.

Esempio:

```
<?php
require_once( TEMPLATEPATH . '/framework/My_Class.php' );

function my_say_hi( $str, $message ) {
$my = new My_Class( $str );
echo $my->metodo( strtoupper( $message ) );
}

?>
```

Quindi nel tema:

```
<?php my_say_hi( 'Ciao', 'WordPress' ); // 'CIAO WORDPRESS' ?>
```

INTERAGIRE: API, ACTION, FILTRI

WordPress mette a disposizioni API, action e filtri per modificare il comportamento del CMS e interagire con esso.
Ad esempio esistono le API per creare gli shortcode, la action wp_enqueue_scripts per inserire file CSS e JavaScript
e il filtro the_content per modificare l'output del contenuto dei post e delle pagine.

A ciascuno di essi possiamo associare il nostro codice
utilizzando le funzioni o i metodi delle classi.

Esempio:

```
function my_paragraph( $atts, $content = null ) {
        extract( shortcode_atts( array(
                'class' => 'special'
           ), $atts ) );
return '<p class="' . $class . '">' . $content . '</p>';
                }

add_shortcode( 'my-paragraph', 'my_paragraph' );
```

Questo codice genera lo shortcode [my-paragraph]. Come potete vedere, WordPress non fa altro che
associare il codice contenuto nella funzione my_paragraph() con il nome dello shortcode.
Quindi tutta la logica del nostro shortcode è racchiusa nella funzione associata.

INTERAGIRE: COME

Notate i due parametri $atts e $content: rappresentano rispettivamente gli attributi e il contenuto dello shortcode.
Questa è una caratteristica comune nelle interazioni con WordPress:
alcune delle funzioni o dei metodi che useremo dovranno avere dei parametri obbligatori:

```
function my_add_author( $content ) {
        if( is_single() ) {
           global $post;

        $author_id = (int) $post->post_author;
        $author_meta = get_user_meta( $author_id );
$written_by = '<p>' . __( 'Scritto da', 'miotema' ) . ' ' . $author_meta['first_name'] . ' ' . $author_meta['last_name'] . '</p>';
        return $content . $written_by;
                }
        return $content;
                }

add_filter( 'the_content', 'my_add_author' );
```

Questo filtro aggiunge il nome e cognome dell'autore del post dopo il contenuto dello stesso solo
se stiamo visualizzando un post singolarmente.

In questo caso il parametro $content rappresenta il contenuto del post ed è obbligatorio: inoltre se la funzione non restituisce
sempre il contenuto non verranno più visualizzati i contenuti dei post.

INTERAGIRE: DETTAGLI

Per usare le API, le action e i filtri è fondamentale
seguire gli esempi proposti nella documentazione
per imparare l'uso di base e una prima implementazione.

Ovviamente la documentazione non può soddisfare tutte le nostre esigenze.
Per questo motivo possiamo effettuare una successiva ricerca su Google per
ottenere subito una risposta ai nostri dubbi.

INTERAGIRE: DETTAGLI

Ad esempio se cerchiamo il filtro pre_get_posts nella documentazione avremo
https://codex.wordpress.org/Plugin_API/Action_Reference/pre_get_posts
e gli esempi forniti coprono molti casi.

Ora immaginiamo di voler filtrare i risultati della ricerca in base a una
variabile GET impostata da noi.
Nella documentazione l'esempio specifico non è presente, quindi dobbiamo effettuare una ricerca
su google.com per "wordpress add query vars".

Approdiamo a questa soluzione:

```
add_filter(  'query_vars', 'my_add_vars'  );

function my_add_vars(  $new_query_vars  ) {
        $new_query_vars[] = 'my-query-var';
        return $new_query_vars;
        }
```

Ora la variabile GET my-query-var è presente tra quelle riconosciute da WordPress.
Quindi possiamo usare l'esempio base della documentazione in questo modo:

```
function my_filter_search( $query ) {
if ( !is_admin() && $query->is_main_query() ) {
        if ( $query->is_search ) {
    $my_var = get_query_var( 'my-query-var' );
    if( isset( $my_var ) && $my_var == '1' ) {
            $cats = array( 67, 65, 66 );
     $query->set( 'cat', implode( ',', $cats ) );
                    }
                }
                }
            }

add_action( 'pre_get_posts','my_filter_search' );
```

HELLO SERVER!

- Conoscere il server su cui operano PHP e WordPress è fondamentale.
- Una buona parte dei problemi che dovrete affrontare sono dovuti alla configurazione del
- vostro server di produzione.

- A voi non interessa diventare sistemisti
- ma solo conoscere ciò che può influenzare PHP e WordPress.

-

HELLO SERVER!

- La prima cosa da sapere è qual'è la versione di PHP in uso.
- WordPress vuole una versione uguale o superiore (meglio superiore)
- alla 5.2.

- Potete verificarlo da soli lanciando un file info.php con il seguente codice:

- `<?php phpinfo(); ?>`

- Con questo saprete tutto sulla vostra installazione di PHP.

- Se l'installazione ha una versione di PHP obsoleta o al limite della compatibilità
- vi consiglio di chiedere un aggiornamento
- o se non è possibile
- un cambio di hosting.

-

HELLO SERVER!

La seconda cosa da conoscere sono le risorse a disposizione di PHP.
Ossia:
memoria (memory_limit)
tempo massimo di esecuzione (max_execution_time)
tempo massimo di input (max_input_time)
valore massimo di upload (upload_max_filesize)

In un mondo ideale questi valori sarebbero accessibili per voi
sia in lettura che in scrittura direttamente nel file php.ini
della vostra installazione di PHP.

Il più delle volte non è così.
Quello che potete fare è provare a modificare questi valori nel file .htaccess
se il vostro hosting lo consente:

```
php_value memory_limit  256M
```

Questo codice vale per il web server Apache. Per nginx o Microsoft dovreste rivolgervi
al supporto tecnico a meno che il pannello di controllo del vostro hosting non abbia questa opzione.

Esistono anche delle soluzioni in PHP da inserire nel file wp-config.php:

```
define( 'WP_MEMORY_LIMIT', '256M' );
```

Le soluzioni in PHP sono efficaci ma non ottimali.
Se siete costretti a ricorrere a quest'ultimo tipo di soluzione
dovreste riconsiderare l'hosting che avete scelto.

HELLO SERVER!

Come scegliere un hosting?
WordPress è basato su PHP + MySQL
e questo non ci aiuta nella scelta dato che
è la combinazione più diffusa in assoluto.

Proviamo a stilare una wish list:

uptime minimo garantito
scalabilità
risorse garantite
soluzioni per la performance (cache, CDN)
pannello di controllo avanzato
gestione degli aggiornamenti di WordPress dal pannello di controllo
monitoring
assistenza garantita
supporto tecnico specializzato anche in WordPress
installazione one-click di WordPress (se non siamo troppo esperti)

Insomma, un hosting che sappia gestire al meglio WordPress!