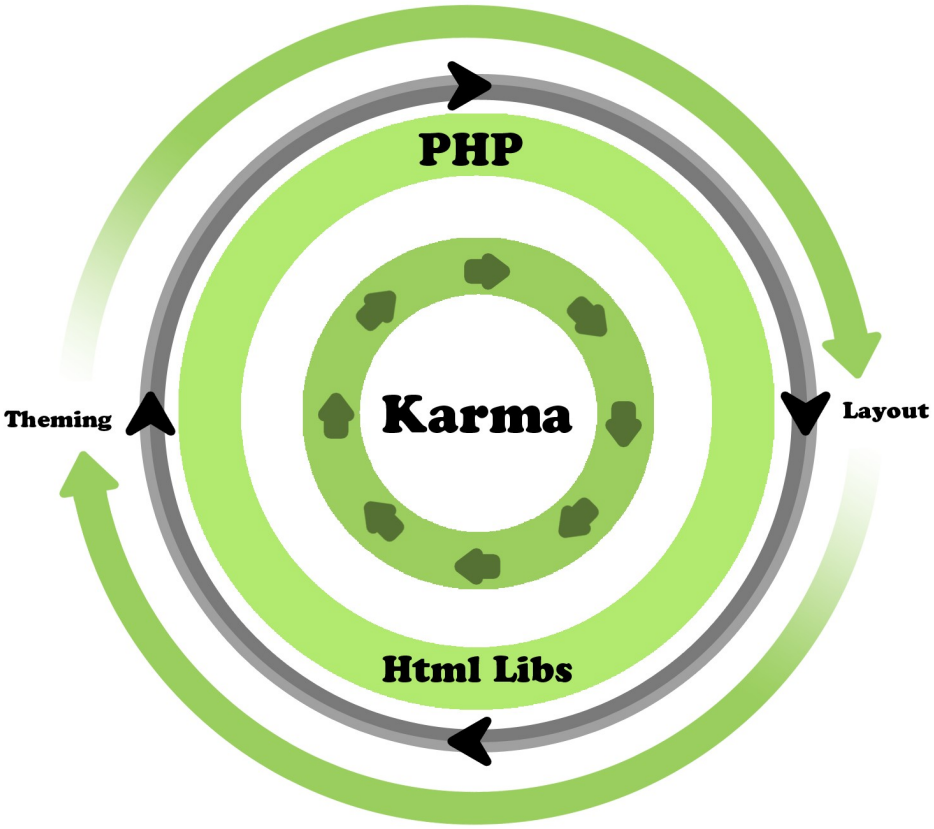
Karma

XHTML/CSS libs - Php
Open Project



Domenico Monaco
kiuz.4ever@gmail.com

Creative Commons 2.5

Karma HTML libs in PHP

Domenico Monaco Marzo 2009

FASE PROGETTUALE

1 Introduzione

Karma è un insieme di librerie php per gestire la formattazione di contenuti provenienti da funzioni php e formattarli secondo i tag html attraverso classi e funzioni appositamente scritte.

Non si presenta come un vero e proprio Template Engine ma piuttosto come un'interfaccia tra funzioni php che generano contenuti e la formattazione HTML orientata verso una possibile gestione da parte di un designer CSS.

Questo vuol dire che si avrà come risultato dall'uso di Karma nelle vostre applicazioni php, contenuti correttamente incapsulati in elmenti XHTML/CSS gestibili in modo ottimale attraverso l'uso dei CSS.

In oltre è implementato un piccolo sistema di riconoscimento e selezione tema.

Il progetto è nato come supporto theming ad un'altro progetto php in cui era eccessivo utilizzare un Template Engine, ma necessitava piuttosto di interfaccia che incapsulasse variabili e contenuti php in tag HTML facilmente gestibili in CSS, ovviamente esso è stato intrapreso anche a scopo didattico personale sviluppando librerie capaci di soddisfare le esigenze tecniche di un Designer che opera principalmente in CSS.

Facciamo un piccolo esempio (i nomi sono completamente fittizi):

- Abbiamo una funzione php che genera una serie di contenuti Es:
  - Nome: Linux
  - Cognome: Gnu
  - Residenza: Italia

Si vuole che i seguenti contenuti debbano essere stampati come una lista, che eventualmente abbia come attributi sufficenti classi e id da poter essere gestiti senza che il programmatore php si preoccupi della loro gestione, forse ignorando o non comprendendo la loro vera utilità da parte del Designer Css.

```
$DettagliUtente = new htmlStruct;

//passiamo guindi le variabili alla funzioni che le stampi sottoforma di lista;

//settiamo le variabili come un array multidimensionale;

$utente=array('Nome'=>'Linux','Cognome'=>'Gnu',');

$DettagliUtente -> printList($utente, 'Eventualmente un titolo alla lista', 'id_tagUL', 'id_tagLI');
```

essa produrrà un risultato simile (questo dipende dalla versione e dall'utilizzo o meno di opzioni aggiuntive descritte dettagliatamente in seguito):

```
<ul id="id_tagUL" class="ulc-0 uli-0">
<h2 class="ID_H_ASSEGNATO-title h-0">TITOLO</h2>
        <li id="id_tagLI" class="lic-0 lii-0">Nome: Linux</li>
        <li id="id_tagLI" class="lic-1 lii-1">Cognome: Gnu</li>
        <li id="id_tagLI" class="lic-0 lii-2">quattro</li>
</ul>
```

Questa viene chiamata Views (vista) di una possibile funzione php, in questo caso ad esempio ViewDettagliUtente(); , che potrà essere utilizzata tranquillamente come si fa in qualsiasi altro Template Engine dei più famosi CMS (Drupal, Wordpress ecc..).

Questo è il risultato. Niente di eccezionale forse, ma se si guarda il risultato in prospettiva di una gestione ottimale attraverso i CSS, questo a mio parere è un ottimo risultato infatti abbiamo come ben si può notare una lista ben abbinata a id e classi gestibili in CSS.

La struttura della pagina rimane completamente a carico delle pagine di template del tema selezionato.

In pratica ho cercato di dare uno strumento php attraverso il quale ci si stacca dall'incapsulamento dei contenuti principalmente orientato alle singole funzioni php, in modo tale che un programmatore php riesca a produrre contenuti gestibili tramite css anche non conoscendo le esigenze del grafico che ci lavorerà in seguito producendo contenuti altamente flessibili utlizzando sintassi che ben conosce, classi e variabili php e nient'altro,

Questo è solo un esempio, le caratteristiche ed i vari utilizzi saranno descritti in seguito.

2 Struttura File e Cartelle

+ **Template**

|-+ **Views**

| |- function-1.view

| |- function-2.view

| |- function-3.view

|

|- load_template.php

|- html.classes.php

+ **Themes**

|- function.php.inc

|- index.php

|- folder.php

|- ... ecc

La cartella template -> è il motore di teming se così si può chiamare, provvede al riconoscimento del tema, generazione viste rese disponibili al Theming che attraverso le pagine di template presenti nella caretella Themes gestirà il layout con inserito all'interno le View delle funzioni.

- **load_template.php** -> si occupa del riconoscimento e selezione del tema indicato nelle impostazioni dell'applicazione php, caricando i file necessari alla visualizzazione dei contenuti. Questo file fa ponte tra:

  applicazione php (attraverso e viste) librerie html tema selezionato

- **html.classes.php** -> è il vero cuore di Karma, esso si presenta come una serie di classi e funzioni destinate ad avere in pasto contenuti richiesti dalle viste delle funzioni php e generare una formattazione ottimale;

La cartella Views -> Contiene le viste di tutte le funzioni che si vuol formattare sottoforma di funzioni che a loro volta sfruttano le librerie html (html.classes.php) per formattare i contenuti, in alternativa sono presenti viste diverse di una stessa funzione.

Prenderà le funzioni php dell'applicazione in input ed estraendo i contenuti sarà possibile creare View per ogni esigenza, utilizzabili nelle pagine di template;

3 Views (viste)

Le **Views** sono le funzioni che producono testo o ipertesto formattato secondo le proprie esigenze attraverso la ricezione in input delle variabili delle diverse funzioni php realizzate senza occuparsi della formattazione dei tag HTML e incapsulare esse in tag XHTML secondo l'esigenza delle **Views** .

Questa è la sezione che più può interessare al programmatore che vuol formattare i propri contenuti senza preoccuparsi di dover aprire, chiudere tag html e dover in'oltre abbinare in modo coplesso classi e id ai tag stessi.

3.1 Come formatta le variabili una View?

Le **View** utlizzano i cosiddetti **Metods** (metodi) per formattare/incapsulare i contenuti, quindi avremo diversi Metods che a loro volta potranno essere utilizzati secono specifiche sintassi.

Ogni Metodo potrebbe avere diverse opzioni di tra cui la possibilità di generare ad esempio div, link, liste puntate, titoli ... ecc.

3.2 Come si opera

1) si crea un file *.view
2) si crea una views creando una semplice funzione che in ingresso potrebbe avere una o più variabili necessarie al funzionamente di una funzione php (nb. non del motore di teming, quelle saranno decise interamente nella view)

   ES.
   function dettagliUtente($utenteVars){
   ...
   .....
   }
3) si crea un oggetto e htmlStruct associando ad esso una variabile
   ES.

   function ViewDdettagliUtente($utenteVars){

   $dettagliUtente = new htmlStruct;

   $dettagliUtente-><METOD_NAME>($utenteVars, [...Options]);

   ...
   .....
   }
4) Eventualmente prima di passare le variabili al metodo potrebbe essere necessario fare delle operazioni secondo le esigenze del Metodo stesso, ad esempio se il metodo supporta o meno la ricezione in input di un array o di una varibile singola;

4 Metods

Verranno qui inseriti di seguito i metodi disponibili e la relativa descrizione di utilizzo per creare una View attraverso essi. Possono essere utilizzati più metodi in una stessa views.

4.1 Basic

Il metodo basic supporta in input o una variabile singola o un array di contenuti, se si preferisce l'array si avrà la possibilità di utilizzare alcune opzioni

- **Variabili Multiple e/o Opzioni di variabili:**
- Con questo metodo è possibile sfruttare il metodo Basic con l'inserimento in input di più variabili sottoforma di un Array Multidimensionale con la possibilità di utilizzare opzioni per ognuna delle variabili inserite, la sintassi è la medesima dell'esempio precedente con l'unica differenza che riceverà in input un array miltidimensionale come indicato di seguito:
- **Sintassi:**
  - $nome-oggetto->basic('<TAG>', '<ID_TAGS>','<CLASS_TAGS>',$VariabiliArrayM);
  - **<TAG>:** div, li, ul, link, hX
  - **<ID_TAGS>:** singolo o array
  - **<CLASS_TAGS>:** singolo o arry
  - **$VariabiliArrayM = Array(**
  - **array('Contenuto','<IF_Link>','<IF_TITLE>','<IF_Bold>');**
  - **array($variabile1,'<IF_Link>','<IF_TITLE>','<IF_Bold>');**
  - **array(.... ....);**
    );
  - **<IF_TITLE>:** contiene un numero da 1->(n) secondo lo standard dei Titolo h1,h2... dei tag HTML, Se 0 o vuoto non viene utilizzato, se maggiore/uguale di 1 applica al contenuto il titolo di dimensione hn;
  - **<IF_Link>:** Se vuoto non applica i tag html del link, se si vuole applicare i tag HTML al contenuto basta inserire i questa locazione la PATH del link e automaticamente verranno generati i tag HTML con associato al percorso la PATH inserita in questa locazione dell'array;
  - **<IF_Bold>:** se si desidera un contenuto in **grassetto**, se 1 viene attiva, se 0 o vuoto non viene attivato il tag grassetto ;

Come si può notare l'unica differenza che i contenuti sono gestiti per riga , cioè ogni riga rappresenta un contenuto che sarà stampato sempre all'interno del metodo Basic , ma che in aggiunta possono avere una o più opzioni attive.

Utilizzo:

```
$utente1= 'Aldo';
$utente->basic('div',$varsID=array('utente'),$varsClass=array('giallo','destra'),$utente1);
```

Risultato:

```
<div id="utente" class="giallo destra">Aldo</div>
```

- **Singola Variabile**

  Sintassi:

  ```
  $nome-oggetto->basic('<TAG>', '<ID_TAGS>','<CLASS_TAGS>',$variabile);
  ```

  <TAG>: div, li, ul, link, hX

  <ID_TAGS>: singolo o array

  <CLASS_TAGS>: singolo o arry

  Utilizzo:

  ```
  $utente1= 'Aldo';
  $utente->basic('div',$varsID=array('utente'),$varsClass=array('giallo','destra'),$utente1);
  ```

  Risultato:

  ```
  <div id="utente" class="giallo destra">Aldo</div>
  ```

```
$variabile2[0]=array('home','index.php','','0','1');
$menu[1]=array('Folder','folder.php','','0','1');
```

```
$ss->genElement('div',$varsID=array('menu'),$varsClass=array('menu-principale','lista'),
$menu);
```

5 In programma:

- Metods
  - **->Basic**
  - list();
  - image();
  - title();
  - parag();
  - bock();
  - link();

- **-> Advanced**
  - iconList();
  - menu();
  - multiList();
  - multiParag();
  - multiBlock();

- **-> Plus**
  - BreadCroumb();
  - region();
  - ajaxBlock();

Other features:

- Resolution Adapter
- Auto Generate Css
- Paginator
- Simple Basic Layout
- Switch Theme