HET 模板引擎使用手册

关于 HET

HET 是一套免费、开源、简单快捷的 PHP 模板引擎，可以实现 PHP 脚本与模板文件分离。

HET 引擎支持模板的编译、缓存、逻辑处理、函数调用等功能，使用方法比较简单，适用于 PHP 程序员与美工日常协作。

最近更新可以从网站 http://www.hellex.cn 获取。

HET 引擎在 PHP4.3.0 及以上版本环境下运行良好。

唯一可能存在的问题有可能是一些虚拟主机不允许使用 mkdir 函数，这种情况下请手动建立所需的编译目录或缓存目录，并将其设为 PHP 可写。

当然如果情况糟糕到连 fopen 函数都无法使用，意味着 HET 也将无法工作。

如果你在实际应用中发现 HET 存在问题，作者敬请你将问题反馈至 hellex@live.com 。

如果有人认为 HET 值得讨论，一个论坛组或论坛版块将会被建立。至少目前看来没有这个需要，并且我们没有找到这样一个角落。

关于这方面的最新消息你也可以从 http://www.hellex.cn 获取。

作者：Hellex
hellex@live.com
2009 年 05 月 23 日

HET 的特性

HET 很快，非常快。
HET 易用，非常易用。
HET 很安全，非常安全。
HET 很强大，非常强大。
……

这似乎有点像广告语了。

在真正使用之前，你可以通过本手册了解 HET 以上的四大特性。

快速入门

PHP

一个快速输出模板的例子：

```
<?php
include 'lib/HET.class.php' ;
$page = new HET ;
$page->out( 'templates/template.html' ) ;
?>
```

逐行解释：

```
include 'lib/HET.class.php' ;
```

引入 HET 引擎，此例中 HET.class.php 与 HETCompiler.class.php 存放在 lib 目录下面。当然，你可以将它们放到其他你想要放的目录，只是不要把它俩分开。

```
$page = new HET ;
```

实例化 HET 引擎，当然你也可以用`$page = new HET();`

```
$page->out( 'templates/template.html' ) ;
```

输出模板函数 HET::out

Out 函数原型 `function out( $template , $data = array() , $compile_dir = '' )`

第一个参数为模板，包括了路径。

第二个参数是要输出的变量数组，可以为空，将在下一章讲到。

第三个参数是混编的模板文件存放的目录，可以为空。

在这里，你需要保证 templates/template.html 这个模板是存在的。否则就会产生如下的错误：

HET->out():templates/template.html does not exist.

至于 templates/template.html 模板要如何设计，我想这就要看美工的了。

如果使用这例子可以正确输出，说明你可以正常使用 HET。

至此，你已经真正使用 HET 了。

页面变量

PHP

我们看一个 usage. php 的简单源码：

```
<?php
include 'lib/HET.class.php'  ;  //引入 HET
$page  = new HET ; //实例化
$data  = array() ;   //页面变量
//页面变量 title 被赋值，在模板里使用{$title}取可输出变量
$data['title']   = 'HET Usage' ;
//数组变量 colors 被赋值，在模板里使用{loop $colors $color}循环输出，并以{/loop}结束循环
$data['colors'] = array( 'Blue' , 'Yellow' , 'Brown' ) ;

$page->out( 'templates/usage.html' , $data ) ;  // 输出模板，第二个参数是页面变量
?>
```

模板

templates/usage.html 模板的源码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
 <head>
  <title>{$title}</title>
 </head>
 <body>
  {loop $colors $color}
    {$color}<br/>
  {/loop}
 </body>
</html>
```

运行结果：

Blue
Yellow
Brown

同时你还可以在浏览器里看到页面的标题是"HET Usage"，正是 usage.php 中设置的值。

设定编译目录

HET 设定编译目录参数有两种方法

PHP

第一种方法：

```php
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;
$page->compile_dir = 'templates_c/usage' ; // 第一设定编译目录方法，在此定义
$data = array() ;
$data['title'] = 'HET Usage' ;
$page->out( 'templates/template.html' , $data ) ;
?>
```

第二种方法：

```php
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;
$data = array() ;
$data['title'] = 'HET Usage' ;
$page->out( 'templates/template.html' , $data , 'templates_c/usage' ) ; // 在此定义
?>
```

第二种方法设定会覆盖第一种方法的设定。
两种方法都可以使用。

HET 会在指定的编译目录下运用自身的算法创建相应的子目录，以避免在同一文件夹下存放过多编译文件的问题。

如果你的环境禁止使用 mkdir 函数，请看下一章“HET 的单目录模式”

单目录模式

如果你的环境不支持 mkdir，你首先需要手动建立编译目录，并且目录设为可写，同时将 HET 设为单目录模式运行。

PHP

HET 设为单目录模式的方法：

```php
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;
$data = array() ;
$data['title'] = 'HET Usage' ;
$page->single_dir = true ;  // 使用单目录模式
$page->out( 'templates/template.html' , $data , 'templates_c/usage' ) ;
?>
```

单目录模式运行的 HET，编译或缓存文件只会保存在指定的目录中。
如果你的环境支持 mkdir，作者并不鼓励你使用单目录模式。
不过如果你想自己在 HET 之外建立更清淅的目录管理机制，这又另当别论，单目录模式是一个不错的选择。

模板基础

模板

本章中所述，全部为模板中的使用方法。

变量输出

语法: {变量名}
只有在 PHP 文件中页面变量中定义的变量才会在模板中起作用

已定义变量：{$title}　　输出：HET Demo
未定义变量：{$name}　　输出：{$name}

变量赋值

语法: {变量名 = 值}

数字赋值：{ $n = 123 }　　输出：{$n}　　结果：123
字串赋值：{ $abc = 'I am abc' }　　输出：{$abc}　　结果：I am abc
变量赋值：{ $ABC = $abc.' too ' }　　输出：{$ABC}　　结果：I am abc too

注册函数赋值：{ $date = date('Y-m-d H:i:s') }　　输出：{$date}　　结果：2009-05-18 18:29:10

数组循环输出

语法:

```
{loop 变量名 输出数组 }
  //循环块
{/loop}
```

例如:

```
{loop $students $std }
  姓名:{$std['name']}
```

```
  年龄:{$std['age']}
  成绩:
        {loop $std['score'] $score } // 嵌套的循环
          {$score['subject']} : {$score['score']}
        {/loop}
 {/loop}
```

输出如下：

姓名:Joey	姓名:Mark	姓名:David
年龄:12	年龄:11	年龄:13
成绩:	成绩:	成绩:
Math : B	Math : A+	Math : B+
English : A	English : A	English : B
Physical : B+	Physical : B	Physical : A+

HET 支持 continue 与 break，请看下一块的逻辑处理的例子。

逻辑处理

语法:

```
{if|elseif|while 条件}
  Statement...
{/if|while}
```

注：条件语体不需要用()括起来, if 或 elseif 开头的以{/if}结束,{while}开始的则以{/while}结束

例如：

```
{$gender = 'female'}  //给$gender 赋值
{if $gender == 'male' }
   He is Male.
{elseif $gender == 'female'}
   She is Female.
{else}
   Unknow.
{/if}
```

输出： She is Female.

使用 while 的例子：

```
{$i=0} //给$i 赋值
{while $i < 10 }
```

```
   {$i = $i+1}
   {$i}
{/while}
输出： 1 2 3 4 5 6 7 8 9 10

使用 continue 的例子(不输出 5) ：
{$i=0}
{while $i < 10 }
   {$i = $i+1}
   {if $i == 5 }{continue}{/if}
   {$i}
{/while}
输出： 1 2 3 4 6 7 8 9 10

使用 break 的例子(不输出 4 以后的) ：
{$i=0}
{while $i < 10 }
   {$i = $i+1}
   {if $i == 5 }{break}{/if}
   {$i}
{/while}
输出： 1 2 3 4
```

使用函数

```
语法: { 函数名() }

 例如：{myfunc()}
 输出：No Param

 在其他语体中使用函数：
 {$date = date('Y-m-d')} // date()函数赋值给$date
 {$date}
 输出：2009-05-18

 带参数使用： {myfunc( $date )}
 输出：2009-05-18

 {$number = 1234.567 }
 {$numberf = number_format($number , 2 )}
 {$numberf}
```

输出：1,234.57

include 的使用

语法：{include("文件名")}
include 的使用依赖于 PHP 程序的注册函数，注册了 include 后即可使用。请使用小写。

例如：{include("included.php")}

结果：Hello World!This is included file.

$_REQUEST 的使用

语法：直接使用，键名不能有单引号
$_REQUEST 的使用依赖于 PHP 程序。

例如：{ $id = $_REQUEST["id"] } 在地址栏上加上?id=123 输出{$id}

结果：123

解释语体

语法：\{ 内容 \}

转义后将输出 { 内容 }，当且仅当需要输出正确的语体时才需要使用转义，转义意味着语体不会被编译。

HET 的解释语体中，在单引号中使用单引号，必须将单引号\转义
例如
{$abc = '$_ABC['abc']'}　错误
{$abc = '$_ABC[\'abc\']'}　正确
{$abc = '$_ABC["abc"]'}　正确
{$dog = 'It's a dog'}　错误
{$dog = 'It\'s a dog'}　正确

务必注意，在语体中使用{}必须将其转义。在语体将{}转义的方法是使用\{ 和\}
比如：{$abc= '\{asdfa\}'} 将会输出结果：{asdfa}

函数的使用

PHP

PHP 中的例子:

```
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;

// 使用 HET::fn()函数注册函数，注册的函数可以在模板中使用
$page->fn ( array( 'date' , 'myfunc' , 'number_format' )  ) ;
$page->fn ( 'money_format' ) ; // 也可以这样单个注册

$page->out( 'templates/template.html' , $data ) ;
?>
```

模板

模板中的例子：

```
语法: { 函数名() }

例如：{myfunc()}
输出：No Param

在其他语体中使用函数：
{$date = date('Y-m-d')} // date()函数赋值给$date
{$date}
输出：2009-05-18

带参数使用： {myfunc( $date )}
输出：2009-05-18

{$number = 1234.567 }
{$numberf = number_format($number , 2 )}
{$numberf}
输出：1,234.57
```

强制重编译

一般来说，HET 生成编译文件后，除非改动模板，否则不会重新编译文件。
在实际应用中，有时候可能会需要手动重编译文件。HET 提供了强制重编译的方法。

PHP

PHP 中的例子:

```php
<?php
include 'lib/HET.class.php'  ; //引入模板类
$page = new HET ; //实例化

// 使用 HET::fc 控制重编译
$page->fc = true ; //开启强制重编译

$page->out( 'templates/template.html' , $data ) ;
?>
```

例子中，PHP 程序将每次都会重新编译一次模板。

使用缓存

在 web 应用中，合理使用缓存能最大限度地提升网站的性能，减轻服务器的压力，缓存技术是 web 应用中一项普遍而重要的技术。

HET 模板引擎提供了简捷的缓存使用方法，并且，它是非常安全的。

PHP

PHP 中的例子：

```
<?php
include 'lib/HET.class.php'  ; //引入模板类

// use_cache 应该在数据处理之前使用，成功使用缓存会产生 exit，省去后面的处理
$page = new HET ;     //实例化
$page->use_cache() ; //紧随实例化，成功使用缓存忽略后面程序

$data = array() ;
//.....
$page->out( 'templates/template.html' , $data ) ;
?>
```

使用缓存(进阶篇)：

```
<?php
include 'lib/HET.class.php'  ;

$page = new HET ;
$page->cache_lifetime = 3600 ;            //缓存生存时间
$page->cache_dir      = 'cache/usage' ;   //缓存存放目录

$page->out( 'templates/template.html' , $data ) ;
?>
```

HET 的缓存安全处理机制：

```
<?php
include 'lib/HET.class.php'  ;

$page = new HET ;
/**
HET 的缓存安全处理机制 ：
```

```
    如果要缓存网址如 abc.com/list.php?id=123&num=456   这样的页面
    将要接受的变量作为 use_cache 的参数传入。
    如$page->use_cache( array('id','num'))
    除了接受的变量外，改变 Query String 的其他部份都不会产生新的缓存，避免被恶意攻击。
**/
$page->use_cache( array( 'id' , 'num' ) ) ;

$data = array() ;
//.....
/**
HET 的缓存安全处理机制 ：
    检查 id 与 page ，  如果检查 id 或 page 不合法，使用 caching 关闭 cache。
    这样就不会产生缓存。适用于不输出页面内容时使用。
**/
$page->caching = false ; // 取消缓存

$page->out( 'templates/template.html' , $data ) ;
?>
```

HET::use_cache 函数的$accept_key 参数是当前页面只接受的变量。

比如一个 abc.com/list.php?id=123&num=456 这样的页面，一般的模板引擎，只要你改变地址的任何一部份，不管是改为 abc.com/list.php?id=123&num=456&asdfasfa121dad 还是其他，它都会产生新的缓存文件，因为不管怎么说，这样看上去肯定会通过参数检查，id 与 num 都是合法的。这样如果有人恶意不停地更改地址，缓存将会越来越多，这可不是好现象。

如果你想避免这种现象，就使用$accept_key 吧。

HET 默认就会为你做这件事。

但如果你认为没有这个必要，你可以告诉 HET 你要取消这种安全机制，如果你确定要这么做的话。

取消 HET 缓存安全处理机制:

```
<?php
include 'lib/HET.class.php'   ;

$page = new HET ;
$page->cache_safe   = false ;   // 取消 HET 缓存安全处理机制

$page->out( 'templates/template.html' , $data ) ;
?>
```

获取结果

有时候你可能需要直接获取页面结果，而不是使用 out 函数输出。比如有些应用需要自主命名生成相应的 html 静态文件。
HET 提供了这样的一个方法。

PHP

获取页面结果:

```php
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;

$data = array() ;
//.....
$result = $page->result( 'templates/template.html' , $data ) ;  //使用 HET::result()获取结果

//将结果生成文件，如静态 html 文件 create_file 是自定义函数
create_file ( $content_filename , $result ) ;
?>
```

灵活应用篇：使用 include

这里举的例子是使用 include，你也可以使用比如 require 等。
不管怎么说，出于安全考虑，都只能是使用 HET::fn 函数注册之后，这些函数或结构语言才能在模板中使用。

PHP

使用 include：

```
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;

$page->fn ( 'include' ) ;  //将 include 当成函数(它其实不是函数)注册即可在模板里使用
$data = array() ;
//.....

$page->out( 'templates/template.html' , $data ) ;
?>
```

模板

include 的使用

```
语法: {include("文件名")}
include 的使用依赖于 PHP 程序的注册函数，注册了 include 后即可使用。请使用小写。

例如：{include("included.php")}

结果：Hello World!This is included file.
```

灵活应用篇：使用$_REQUEST

使用$_REQUEST**：**

```
<?php
include 'lib/HET.class.php'  ;
$page = new HET ;

$data = array() ;
$data['_REQUEST'] = $_REQUEST ;  // 这样便可在模板里使用 $_REQUEST 数组变量
//.....

$page->out( 'templates/template.html' , $data ) ;
?>
```

模板

$_REQUEST 的使用

```
语法：直接使用，键名不能有单引号
$_REQUEST 的使用依赖于 PHP 程序。

例如：{ $id = $_REQUEST["id"] } 在地址栏上加上?id=123 输出{$id}

结果：123
```

(全文完)