HDPHP 开源框架 2014.12

开发代号 : 盾友

HDPHP 框架是 100% 自由的，您可以尽情享受和分享

为友爱而生

后盾网 人人做后盾

修改时间 2014 年 12 月 17 日 上午 1 时 1 分 HDPHP 框架 (版本以年月构成)

HDPHP 敏捷型框架产品

作者：向军

HDPHP 框架为友爱而生

任何人都可以加入和扩展 HDPHP 框架，让更多人通过 HDPHP 框架了解到你 .....

后盾网 www.houdunwang.com　HDPHP 官网 www.hdphp.com

社区开发团队　等待你的加入 .....

目录

1. HDPHP 框架只为让你快速开发

后盾 HDPHP 框架是一个为用 PHP 程序语言编写网络应用程序的人员提供的软件包。提供强大的、完整的类库包，满足开发中的项目需求，HDPHP 框架可以将需要完成的任务代码量最小化，大大提高项目开发效率与质量，当然使用是非常简便、快捷的。高效的核心编译处理机制让系统运行更快，提供丰富的的错误解决方案，让修正代码变得更快速。

做为优秀的框架产品，HDPHP 在系统性能上做了大量的优化处理，只为让程序员使用 HDPHP 框架强悍的功能，用最短的时间完成项目的开发。为了便于快速进行项目开发，框架也提供了相应的前端功能组件使程序员从繁琐的组件调试中解脱出来。

HDPHP 框架定位：简单快速学习，简单快速开发网站。

1_1 运行环境

PHP 版本要求 5.16 或更高版本

可以运行在 UNIX、LINUX、WINDOWS、苹果 Macintosh 电脑等平台

windows 环境下推荐使用 wamp 集成化环境进行学习使用

1_2 如何获得 HDPHP 框架

获得 HDPHP 的方式很简单只要登录 www.hdphp.com 下载即可，你完全可以免费使用，在法律允许情况下可以用于任何用途，包括任何商业产品的开发。

1_3 获得帮助

a. 登录 hdphp.com 官网获得帮助

b. 通过官方邮件获得帮助 2300071698@qq.com

c. 登录 bbs.houdunwang.com 论坛参与讨论

通过论坛解决问题的途径是最快速的，因为你的问题也有可能别人已经提出过，所以你可以先通过论坛获得帮助，这样效果最快。

2. 许可协议

2_1 遵循协议

HDPHP 遵循 Apache2 开源协议发布，并提供免费使用。

Apache Licence 是著名的非盈利开源组织 Apache 采用的协议。该协议和 BSD 类似，

同样鼓励代码共享和尊重原作者的著作权，同样允许代码修改，再发布（作为开源或商业软件）。需要满足的条件也和 BSD 类似：

1. 需要给代码的用户一份 Apache Licence

2. 如果你修改了代码，需要在被修改的文件中说明。

3. 在延伸的代码中（修改和有源代码衍生的代码中）需要带有原来代码中的协议，商标，专利声明和其他原来作者规定需要包含的说明。

4. 如果再发布的产品中包含一个 Notice 文件，则在 Notice 文件中需要带有 Apache Licence。你可以在 Notice 中增加自己的许可，但不可以表现为对 Apache Licence 构成更改。

Apache Licence 也是对商业应用友好的许可。使用者也可以在需要的时候修改代码来满足需要并作为开源或商业产品发布 / 销售。

具体协议参考：http://www.apache.org/licenses/LICENSE-2.0.html

2_2 许可方式

只要符合以下条件，你将被允许使用、复制、修改以及分发本软件和它相关的文档，包括你可以修改或者不修改地用于任何目的：

这个许可协议的一份拷贝必须包含在分发的软件中。

再分发源代码时必须在所有源代码文件中保留上方的版权提醒。

任何修改过的文件必须加上对原始代码修改的注释以及修改者名称。

任何由本软件衍生的产品必须在它们的文档以及 / 或者随分发提供的物品中表明它们来源于后盾网。

除非事先得到后盾网的书面许可，否则任何本软件的衍生产品都不得叫做 'HDPHP 框

架 '，也不得在名称中出现 'HDPHP'。

2_3 赔偿

你必须认可对任何因使用或者误用本软件或者违反任何本许可协议条款所产生的直接、间接、附带的或相应的第三者索赔、诉讼费用承担赔偿 ,皆与本软件的作者和任何贡献者无关。

2_4 无担保声明

该软件仅以实际效果为准 ,不做任何明示或暗示的保证 ,包括但不限于保证的质量 ,性能，不侵权，适销性或适用于特定用途。

2_5 责任限制

你将承担所有安装和使用本软件的风险。

3. HDPHP 框架特性

3_1 免费

HDPHP 是经过 Apache/BSD-style 开源许可授权的，只要你愿意就可以免费使用它。同时后盾网 www.houdunwang.com 提供强大的技术支持，记住这一切都是免费的。

3_2 轻量级

真正的轻量级。我们的核心系统只需要一些非常小的库，这与那些需要更多资源的框架完全相反。额外的库文件只在请求的时候加载 ,依需求而定 ,所以核心系统是非常快而且轻的。

3_3 快速

速度非常快。HDPHP 框架提供多项优化策略完善的功能处理类，你要找到一个比 HDPHP 表现更优的框架应该很难吧。

3_4 采用 MVC 设计模式

HDPHP 使用了模型（Model）- 视图（View）- 控制器（Controller）的方法，这样可以更好地使表现层和逻辑层分离。这对项目的模板设计者来说是非常有用的，它最小化了模板中的程序代码量。

使用 MVC 的目的是将 M 和 V 的实现代码分离，从而使同一个程序可以使用不同的表现形式。比如一批统计数据你可以分别用柱状图、饼图来表示。C 存在的目的则是确保 M 和 V 的同步，一旦 M 改变，V 应该同步更新。

3_5 生成干净的 URL

HDPHP 生成的 URL 非常干净而且是对搜索引擎友好化的。不同于标准的 ' 字符串查询 ' 方法，HDPHP 使用了基于段的 PATHINFO 方法，同时提供强大的 URL 路由功能。

3_6 功能强大

HDPHP 框架拥有全范围的类库，可以完成大多数通常需要的网络开发任务，包括： 读取数据库、发送电子邮件、数据确认、保存 session 、对图片的操作，以及支持 XML-RPC 数据传输等。

3_7 可扩展的

这个系统可以非常简单的通过自定义类库、辅助函数来进行扩展，或者也可以通过扩展类、系统钩子来实现。

3_8 对象关系映射（ORM）

对象 - 关系映射（Object/Relation Mapping，简称 ORM），是随着面向对象的软件开发方法发展而产生的。面向对象的开发方法是当今企业级应用开发环境中的主流开发方法，关系数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，对象 - 关系映射 (ORM) 系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

面向对象是从软件工程基本原则（如耦合、聚合、封装）的基础上发展起来的，而关系数据库则是从数学理论发展而来的，两套理论存在显著的区别。为了解决这个不匹配的现象，对象关系映射技术应运而生。

3_9 别名

别名就是指这个函数的另外一个使用方法，如 del 用于更新记录，他拥有一个别名是 delete，del 与 delete 使用方法一模一样只是不同的名称而已。HDPHP 框架为更大的适 用性引用别名概念，支持函数或类方法多个不同名称。

4. 目录结构

目录	说明
Extend	扩展目录
Extend/Org	扩展应用包
Extend/Tool	集成工具包
Data	静态数据目录
Config	基本配置文件
Lib	核心核心目录
Lib/Core	核心文件
Lib/Driver/Cache	缓存驱动
Lib/Driver/Db	数据驱动库
Lib/Driver/Model	数据模型库
Lib/Driver/Session	Session 驱动库
Lib/View	视图驱动库
Lib/Functions	函数库
Lib/Language	语言包
Lib/Tpl	框架模板目录

5. GET 私有变量

GET 私有变量指 HDPHP 占有的 $_GET 值，开发人员请不要使用这些变量，否则会产生异常。

方法名	说明
$_GET['g']	模块组
$_GET['m']	模块名
$_GET['c']	控制器名
$_GET['a']	动作名
$_GET['page']	分页页码变量

6. 什么是 MAC

MAC 定义是 HDPHP 框架的特有定义，如果你以前使用过别的产品，可能会感觉比较陌生，不过 MAC 的操作方式，会让网站的规划更有条理，扩展性更强，下面我们来了解一下 MAC 的具体概念。

M Module 模块

我们可以把网站中的后台 admin 当成一个模块，前台会员中心 member 当成一个模块，有了模块的概念，网站的扩展就会变的很轻松了。

C Controller 控制器

比如说网站的后台模块 admin 中有用户管理控制器、权限分配控制器、用户登录控制器、友情链接管理控制器等等，这些都是控制器。

A Action 动作

比如说文章管理模块包扩文章的添加、修改、删除等动作 ,用户登录模块包括验证码显示，用户验证等动作，这些就是动作。

7. 单入口文件

HDPHP 框架采用单入口文件访问策略，无论从安全性还是方法调用及文件加载方面都带来了很高的便捷性。

7_1 入口文件常量介绍

APP_PATH	应用目录
DIR_SAFE	是否创建目录安全文件
DEBUG	调试模式
DEBUG_TOOL	显示 DEBUG 面板
TEMP_PATH	Temp 目录
TEMP_FILE	编译文件名
MODULE_LIST	批量创建模块

7_2 创建单入口文件

最简单设置

```
define('DEBUG',True);
define('APP_PATH','Admin/');
define('MODULE'_LIST,',Index,Admin');
require 'hdphp/hdphp.php';// 框架主文件
```

注意

a. APP_PATH 必须以斜杠结尾

b. 默认会创建目录安装文件 index.html

7_3 核心编译文件

框架会将核心文件进行编译生成 ~Boot.php 文件，减少 IO 开销，加快运行速度。

不生成编译文件

单入口文件中添加 define('DEBUG',FALSE)， 就不会生成核心编译文件。

定义编译文件名

```
define('TEMP_FILE','~runtime.php');// 核心编译文件
```

```
require './hdphp/hdphp.php'; // 框架主文件
```

7_4 DEBUG 开发模式

特点

a. 不缓存表字段，表结构修改立刻生效

b. 不创建 ~Boot.php 编译文件

c. 每次模板的修改立刻生效

d. 提供丰富的调试工具，类似火狐的 Firebug 工具

8. 9. 配置文件

框架采用多级配置设置。级别由低至高排序如下：

C() 函数 > 模块配置 > 应用配置 > 框架基本配置

8_1 框架核心配置项

框架的 Config 目录中，程序员不用修改框架核心配置文件。

8_2 应用配置

应用下的 Common/config/config.php 文件，应用配置会作用于所有模块。

8_3 模块配置项

模块下的 Config/config.php 文件，只能作用于本模块。

9. 控制器 Controller

控制器是视图与模型间的中间层，起到中间调度作用。控制器收到视图反馈信息， 决定是否交于模型处理，之后将结果返回视图层，完成一个交互流程。

9_1 控制器文件命名规范

文件名由控制器名 +Controller+.class.php 构成，例如 UserController 控制器就要创建成 UserController.class.php 文件

9_2 __init 构造函数

__init 函数是动作执行前自动执行的特殊方法，用于完成初始化操作。

9_3 EmptyController 控制器

访问的控制器不存在时，自动执行的控制器。

9_4 __empty 空动作

当访问的动作不存在时，自动执行 __empty 动作

10. 自动加载

自动加载是用于加载文件扩展功能 ，自动加载需要设置 AUTO_LOAD_FILE 配置项。

配置项设置

'AUTO_LOAD_FILE' => array('Hdcms/Lib/functions.php','hdphp.php')

加载 Hdcms/Lib/functions.php 文件与模块 Lib 目录下的 hdphp.php 文件

注意

a. 必须添加扩展名 .php

b. 应用或模型 Lib 目录下创建的文件 , 不用添加路径

11. import 导入类库

import 函数用于导入类文件。

语法：import($class=null,$base=null,$ext='.class.php')

参数	说明
$class	类名
$base	目录
$ext	类扩展名

导入模块 Lib 目录下的 Controller.class.php

import('Lib.Controller') 或 import('Controller')

目录导入模块 Tag/Html.class.php

import('Tag.Html')

导入应用 Class 目录下的 Html.class.php 文件

import('@.Class.Html')

导入 Member 模块的 HtmlController.class.php

import('Member.Controller.HtmlController')

导入框架 Extend/Tool 目录下 Html.class.php 文件

import('HDPHP.Extend.Tool.Html')

导入网站根目录下 Model 目录中的 RbacModel.class.php 文件

import('RbacModel','Model');

导入当前应用 Common/Lib 目录下的 Auth.inc.php 文件

import('@.Common.Lib.Auth',NULL,'.inc.php')

12. 钩子 Hook

HDPHP 框架提供了钩子处理机制，就是在某一个时机自动执行某些功能。

配置钩子

a. 定义 Hook 配置项

b. 配置可以定义在应用或模块 config.php 文件中

钩子处理类

a. 一个钩子可以指定多个处理类

b. 处理类可以定义在应用或模块的 Hook 目录

c. 处理类的后缀为 Hook.class.php

d. 如果应用与模块有同名钩子时，只会执行模块的钩子

e. 处理类必须有 public function run(&$param) 方法

12_1 系统钩子

钩子名称	说明
APP_INIT	应用初始化
APP_BEGIN	应用运行前
APP_END	应用运行后
CONTROLLER_START	模块执行前
CONTROLLER_END	模块执行后
VIEW_START	模板显示前 (display 方法执行前)
VIEW_END	模板显示后 (display 方法执行后)

12_2 定义钩子

首先设置配置项 Hook，然后创建钩子处理类。

实例

配置 Hook

```
1  'Hook'=> array(
```

```
2     'APP_INIT'=>array('LoginHook')
3   )
```

处理类

处理类序必须定义在 Hook 目录中，必须以 Hook.class.php 结尾

```
1   class HeaderHook{
2   public function run(&$options){
3     header('Content-type:text/html;charset=utf-8');
4         echo ' 设置字符集 ';
5     }
6   }
```

因为设置了 APP_INIT 钩子，在应用开始时会自动执行钩子处理程序。

12_3 Hook::listen

使用 Hook::listen() 方法来执行定义好的钩子的所有处理类。

执行 USER_VALID 钩子

```
Hook::listen('USER_VALID')
```

12_4 Hook::exe

使用 Hook::exe( ) 方法执行一个钩子处理类

执行 LoginHook.class.php 处理程序类

```
Hook::exe('LoginHook')
```

13. 插件

Addon 与 Hook 使用方法基本一致，不同点在代码存放目录与配置项设置。

注意

a. 插件必须存放在 Addon 目录中

b. 插件类必须以 Addon.class.php 结尾

c. 在入口文件定义 define('APP_ADDON_PATH','Addons/') 可改变插件目录

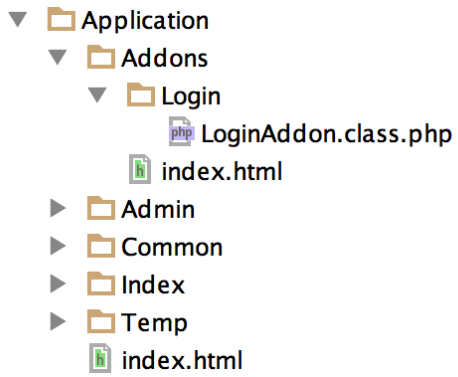
13_1 基本使用

配置项

配置项时不要加 Hook 后缀，这与 Hook 配置不同

```
1 return array(
2   'Hook' => array(
3     'Test' => array('Login')
4   )
5 );
```

目录结构



插件主文件

```
1 class LoginAddon
2 {
3   public function Test()
4   {
```

```
5        echo ' 插件执行 ..';
6    }
7 }
```

在控制器中调用插件

```
1 class IndexController extends Controller
2 {
3    public function index()
4    {
5        Hook::listen('Test');
6    }
7 }
```

13_2 插件模块

其实插件就是一个模块，可以像一般模块一样在插件目录中创建 Controller、Model、View、Config 等目录，使用方法与普通模块一样。

即然插件就是一个模块，那么我们也可以直接访问插件控制器，URL 地址与普通模块相似，只是加上 g 参数即可。

访问 Login 插件中的 AdminController 控制器

```
g=Addons&m=Login&c=Admin&a=index
```

14. Hook 钩子类

Addon 与 Hook 的操作都是使用 Hook.class.php 类完成的，下面介绍 Hook 类的内 部方法。

add 添加钩子

static public function add($hook, $action)
@param $hook 钩子名称

@param $action 处理类

get 获得钩子信息

static public function get($hook = '')
@param string $hook 钩子名

import 批量导入钩子

static public function import($data, $recursive = true)
@param $data 钩子数据

@param bool $recursive 是否递归合并

listen 监听钩子

static public function listen($hook, &$param = null)
@param $hook 钩子名

@param null $param 参数

exe 执行钩子

static public function exe($name, $action, &$param = null)
@param $name 钩子名

@param $action 钩子方法

@param null $param 参数

15. URL 访问方式

HDPHP 框架支持丰富的 URL 访问方式，可以根据应用场景选择合适的 url 访问规则。

控制器命名

控制器命名必须使用 pascal 命名规范，即单词首字母大写，如 UserController.class.php，在 URL 中使用 User 访问即可

控制器名称中含有多个单词如 HdphpUserController.class.php，在 URL 中使用 c=hdphp_user 或 c=HdphpUser 两种形式访问

受影响的配置项

```
1  'HTTPS'              => FALSE,    // 基于 https 协议
2  'URL_REWRITE'        => 0,        //url 重写
3  'URL_TYPE'           => 1,        // 1:PATHINFO  2: 普通模式 3: 兼容模式
4  'PATHINFO_DLI'       => '/',      //PATHINFO 分隔符
5  'PATHINFO_VAR'       => 'q',      // 兼容模式 get 变量
6  'HTML_SUFFIX'        => '',       // 伪静态扩展名
7  'VAR_GROUP'          => 'g',      // 模块组 URL 变量
8  'VAR_MODULE'         => 'm',      // 模块变量名
9  'VAR_CONTROLLER'     => 'c',      // 控制器变量
10 'VAR_ACTION'         => 'a',      // 动作变量
11 'DEFAULT_MODULE'     => 'Index',  // 默认模块
12 'DEFAULT_CONTROLLER' => 'Index',  // 默认控制器
13 'DEFAULT_ACTION'     => 'index',  // 默认方法
14 'CONTROLLER_FIX'     => 'Controller',// 控制器文件后缀
```

默认动作

http://localhost/index.php

访问配置项指定的默认模块、默认控制器、默认动作

访问具体动作

index.php?m=Admin&c=User&a=admin

访问 Admin 模块 User 控制器的 admin 动作

15_1 普通访问方式

普通 GET 方式访问

index.php?m=Admin&c=User&m=add

访问 Admin 模块 User 控制器的 add

15_2 pathinfo 访问

pathinfo 模式提供了更佳的 seo 优化，同时 url 地址更精巧。

访问 Admin 模块 User 控制器中的 show 方法

index.php/Admin/User/show

访问 Admin 模块 User 控制器中的 show 方法并传递参数 uid=2

index.php/Admin/User/show/uid/2

设置分隔符

需要设置 PATHINFO_DLI 配置项

index.php/Index_Index_bb.html

伪静态模式

index.php/Admin/User/add.html

访问 Admin 模块 User 控制器的 add 动作

15_3 兼容模式访问方式

如果不支持 pathinfo 访问规则，可以使用兼容模式，需要修改以下几个配置：

```
1 URL_TYPE=>3
2 PATHINFO_VAR=>'q'
```

访问 Admin 模块 User 控制器中的 add 动作

index.php?q=Admin/User/add

16. U 函数

U 函数可以根据 URL 配置项生成规范的 URL，可以让整个网站 URL 自动适应，同时 U 函数会按路由规则进行解析处理 。

语法：U($pathinfo, $args = array())

设置方法

```
U('add');
```

设置控制器 / 方法

```
U('Channel/add')
```

设置模块 / 控制器 / 方法

```
U('Admin/Channel/add')
```

第三个开始为参数

```
U('Admin/Channel/add/cid/1/name/hdphp')
```

传字符串参数

```
U('channel/add','name=houdunwang&uid=1')
```

传数组传参

```
U('channel/add',array('name'=>'houdunwang','uid'=>1))
```

普通路由规则转换

需要将 URL_TYPE 配置设置为 1

```
'route' => array(
    'info/:city/:row'=>'index/index'
)
U('index','city=beijing&row=200');
```

结果为：info/beijing/200

正则表达式路由规则转换

需要将 URL_TYPE 配置设置为 1

```
'route' =>  array(
```

```
    '/^([a-z]+?)_(\d+)$/'=>'Index/Index/show/city/#1/row/#2'
)
U('Index/Index/show','city=beijing&row=200');
```

生成的 URL 为 index.php/beijing_168

普通模式正则路由规则转换

需要将 URL_TYPE 配置设置为 2

```
'route' => array(
    ':city_:row'=>'c=index&m=index'
)
U('index', 'city=beijing&row=168');
```

生成的 URL 为 index.php?beijing_168

17. 获得系统数据 Q 函数

Q 函数是对 $_GET、$_POST、$_SESSION、$_REQUEST、$_COOKIE、$_GLOBALS、$_SERVER 等超全局数组操作的函数。

Q($var, $default = null,$filter = null)

说明

a. 过滤函数默认使用配置项 FILTER_FUNCTION 的设置

b. 变量是引用类型传参，所以会改变全局变量

c. 不指定数据类型，默认操作 $_REQUEST

参数	说明
$var	要处理的变量（引用类型）
$default	当变量不存在时的默认值
$filter	过滤的函数，当为 null 时不进行顾虑函数处理，当为 '' 时使用配置项 C('fieldFilterTION') 中的函数进行过滤处理

读取 $_REQUEST['cid'];

```
Q('cid');
```

获得所有 $_GET 数据

```
Q('get.');
```

获得所有 $_POST 数据

```
Q('post.');
```

获得 $_POST['webname'] 并执行 htmlspecialchars 与 strtoupper 函数

```
Q('post.webname',NULL,array('htmlspecialchars','strtoupper'));
```

获得 $_POST['webname'], 当变量不存时，设置值为 ' 后盾网 '

```
Q('post.webname',' 后盾网 ');
```

获得 $_SESSION['uid'] 值

```
Q('session.uid',NULL,'intval');
```

获得 $_COOKIE['cart'] 值

```
Q('cookie.cart');
```

18. 判断请求类型

通过判断请求类型，我们可以达到以下几点好处：

a. 根据不同请求类型做出不同的业务处理

b. 同时也过滤掉不安全的请求，增强网站的安全性

常量

通过框架提供的常量快速判断当前请求的类型

常量	说明
IS_GET	get 请求
IS_POST	post 请求
IS_PUT	head 请求
IS_DELETE	delete 请求
IS_AJAX	异步 AJAX 请求

代码演示

```
1  class indexController extends Controller{
2      function add_user(){
3          if(IS_POST){
4              $db = M('news');
5              $db->add();
6          }else{
7              $this->error(' 请求非法 ');
8          }
9      }
10 }
```

19. URL 路由器

URL 路由器可隐藏网站物理文件结构提高安全性，美化 URL 地址便于 SEO。建议使用 U() 函数生成 url, 会根据路由规则生成 url 地址。

19_1 普通路由

无参数路由

配置项

```
1 'route'=>array(
2   'user/:uid'=>'Index/Index/user'
3 )
```

访问 Index 模块 Index 控制器的 user 动作

http://localhost/index.php/user/8

有参数路由

配置项

```
1 'route'=>array(
2 'index_:id'=>'Admin/Channel/show'
3 )
```

访问 Admin/Channel/show，并传递参数 id=8

http://localhost/index.php/index_8

19_2 正则路由

正则路由规则可以实现更加灵活的 URL 定制

修改配置项

```
1 'route'=>array(
2   '/^user(\d+)$/'=>'index/user/uid/#1'
3 )
```

访问 user 动作并传递参数 uid=6

http://localhost/index.php/user6

19_3 支持 QUERY 方式路由

HDPHP 框架路由不仅支持 PATHINFO 方式同时支持 GET 方式

配置项

```
1 'route' => array(
2     '/^([a-z]+?)_(\d+)/'=>'m=Index/c=Index&a=Index&city=#1&row=#2'
3 )
```

访问 Index/Index/show 并传递参数 city=beijing 与 row=20

http://localhost/hdcms/index.php?beijing_20

20. url 伪静态

url 伪静态处理易于被搜索引擎收录，同时隐藏了原始 url 所以更加安全。

url 伪静态需要隐藏项目入口文件，以及定义相应的路由规则，这样才可以生成完美的伪静态，路由的使用在上面已经进行了介绍，下面来学习隐藏项目入口文件。

20_1 隐藏项目入口文件

服务器 rewrite 配置

a. 将 Apache 配置文件 httpd.conf 中加载 mod_rewrite.so 模块

b. 修改配置 httpd.conf 中的 AllowOverride Node 为 AllowOverride All 使 Apache

c. 支持 .htaccess 文件

d. 在入口文件同级目录中创建 .htaccess 文件

.htaccess 内容设置

```
<IFModule mod_rewrite.c>
# 开启 URL 重写功能
RewriteEngine On
# 请求内容不是目录
RewriteCond %{REQUEST_FILENAME} !-d
# 请求内容不是文件
RewriteCond %{REQUEST_FILENAME} !-f
# 重写 URL 规则
RewriteRule  ^(.*)$ index.php\/$1 [L]
</IFModule>
```

隐藏代码中的入口文件

修改配置 'URL_REWRITE'=>true，这样通过 U() 函数生成的 url 会去掉入口文件。

21. success 与 error

success 与 error 方法都是 Controller 基类控制器的方法，success 用于返回成功信息,error 方法用于返回错误信息，默认会调用 View/Public 目录中的同名模板文件。

如果请求是异步 (Ajax) 提交的，系统会自动返回 JSON 数据，如执行 success() 方法会返回 array('status'=>1,'message'=>' 提示信息 ') 数组解析后的 JSON 数据。

success($message = ' 操作成功 ', $url = NULL, $time = 1, $tpl = null)

参数	说明
$message	提示信息
$url	跳转地址（默认回调历史记录）
$time	跳转时间
$tpl	模板文件默认为配置项 TPL_SUCCESS 或 TPL_ERROR 的值

22. Ajax 返回数据

HDPHP 框架提供通过 ajax 返回 Ajax 请求数据，使用方法非常简单。

$this->ajax($data, $type = 'JSON')

参数	说明
$data	异步返回的数据
$type	返回的数据类型，包含 JSON(默认),XML,TEXT

```
1 $data = array('webname' => ' 后盾网 ', 'url' => 'http://hdphp.com');
2 $this->ajax($data, 'JSON')
```

当通过 Ajax 异步请求 index 方法时，系统会返回 JSON 数据给客户端，当然可以指定不同的数据类型如 XML 返回给客户端。

23. 模型

模型是专门用来和数据库打交道的。例如，假设你想用 HDPHP 框架制作一个文章系统，那么你需要通过模型类来完成操作，与数据库操作的所有方法 HDPHP 框架已经封装到的模型的内部，你可以完成插入、更新、删除数据的操作，当然还有更多高级功能。

特点

可以对基础模型类进行随意的扩充

支持加载其他模块或其他应用类某个模块的模型

丰富的加载模型方式

功能强大的数据操作方式

自动验证

自动完成

自动映射

表单令牌

.....

23_1 基本模型

M($table = null, $full = null,$param=null,$driver=null)

参数说明	说明
table	数据表
$full	不添加表前缀
$param	附加参数 (M 函数无效)
$driver	驱动程序

实例化 User 表模型

```
$db = M('User')
```

不添加表前缀

```
$db = M('user',true)
```

23_2 扩展模型（K 函数）

HDPHP 框架支持模型功能的扩展，扩展模型需要继续基本模型或高级模型。扩展模型要放在应用或模块的 Model 目录中。

K($model, $full,$param = array(),$driver=null)

参数	说明
model	扩展模型名称
full	不添加表前缀
param	附加参数
driver	驱动程序

扩展模型

```
1 class UserModel extends Model{
2   public $table='user';// 设置表名
3     function __init($param){
4       p($param);
5     }
6 }
```

控制器代码

```
K('user',false,array('cid'=>1,'webname'=>'houdunwang.com')
```

24. CURD

HDPHP 框架的 CURD 的自由度超过你的想象，当开启 DEBUG 模式后，所有 SQL 语句均会在调试窗口显示，一目了然。

24_1 find 单条查询

通过 find 操作进行简单的单条查询操作。

查找主键值为 18 的记录

```
$db->find(18);
```

24_2 all 查询（别名 select）

通过 all 查找所有记录数 。

查找表中所有记录

```
$list = M('news')->all();
```

24_3 table 临时改变表

通过 table() 方法可快速切换操作表

改变操作表为 user

```
M('news')->table('user')->all();
```

24_4 join 表关联

使用 join() 可以快速实现表间关联

关联 user 表与 role 表

```
$data =M()->join('__user__ u JOIN __role__ r ON u.rid=r.rid')->all();
```

24_5 max 查找最大的值

查找年龄最大的用户

```
M('user')->max('age');
```

24_6 min 查找最小的值

查找最便宜的商品

```
M('goods')->min('price');
```

24_7 avg 求平均值

查找商品平均价格

```
M('user')->avg('price');
```

24_8 sum 求和

获得点击数之和

```
M('news')->sum('click')
```

24_9 count 统计操作

统计会员总数

```
M('user')->count()
```

24_10 field 字段集

返回字段为 uid 与 username

```
$db->field('uid,username')->select();
```

以数组形式传参

```
$db->field(array('uid','username'))->select();
```

更灵活的传参

```
$db->field(array('uid','concat(username,"-",uid)'=>'userid','LEFT(username,7)'=>'name'))- >select();
```

SELECT uid,concat(username,"-",uid) AS userid,LEFT(username,7) AS name FROM hd_user

字段排除

获得除 content,title 以外的所有字段

M('news')->field(array('title','content'),true)->all();

24_11 limit 取部分数据

limit 方法就是为了生成 SQL 的 limit 部分

查找第 2 条记录开始的 5 条记录

$db->limit('2,5')->all();

24_12 排序 ORDER

按 id 从大到小排序

$db->order('id desc')->all();

24_13 getField 获得指定字段值

为了使查询更加灵活，HDPHP 框架提供了 getField 按字段名获得结果的方法

获取唯一值

$db->where('id=32')->getField('title');

无论结果有多少个只返回一个值

满足条件记录的所有 title 字段

$db->where('id>2')->getField('title',true);

两个字段列表

$db->getField('id,title');

两个字段时返回一维数组，第一个字段做为键名使用，第 2 个字段做为键值

多个字段时

```
$db->getField('id,title,click');
```

多个字段返回二维数组，第一个字段值做为键名使用，其余字段做为键值

24_14 GROUP 分组操作

HDPHP 框架提供了完善的分组操作方法，自由指定分组参数使发送 SQL 更容易

按 id 与 name 分组查询

```
$list =$db->group('id,name')->all();
```

```
SELECT `id`,`name`,`sendtime` FROM hd_demo GROUP BY id,name
```

24_15 HAVING 分组条件

在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与合计函数一起使用，所以使用 HAVING 对分组进行条件筛选，所以在使用 HAVING 时应该使用 group 分组。

获得记录条数大于 2 的 cid 值

```
M('news')->having('count(*) >2')->group('cid')->select();
```

24_16 add 添加数据（别名 insert）

特点：

自动过滤非法字段

自动对插入数据进行安全处理

没有传入参数时使用 $_POST 值

返回值为新增主键值或 true

实例

```
1 $data=array('uname'=>' 后盾网 ','url'=>'http://www.houdunwang.com');
2 $lastId = $db->add($data)
```

ORM 属性映射

```
1 $db->username = ' 李四 ';
```

```
2 $db->web = 'houdunwang.com';
3 $db->add();
```

24_17 replace 添加数据

数据中存在主键则更新否则添加数据。

```
1 $data =array('id'=>1,'name'=>' 后盾网 ');
2 M('news')->replace($data);
```

24_18 save 更新（别名 update）

特点

a. 自动过滤掉非法字段

b. 自动进行数据安全处理

c. 默认以 $_POST 数据更新

d. 必须有更新条件，防止误更新

e. 如果参数中存在主键值将以这个值为条件进行更新数据

uid 为表主键，使用数据中的主键为条件进行更新

```
1 $data=array('uid'=>9,'username'=>' 郭富城 ');
2 $db->save($data);
```

24_19 inc 增加值

将 id 为 4 记录的 total 加 1

```
M('news')->inc('total','id=4',1);
```

```
update hd_shop set total=total+1 where id=4
```

24_20 dec 减少值

将 id 为 4 记录的 total 减 1

```
M('news')->dec('total','id=4',1);
```

update hd_shop set total=total-1 where id=4

24_21 del 删除（别名 delete）

为了屏蔽误删除 del 方法必须指定条件

删除主键值为 58 的数据

M('news')->del(58);

DELETE FROM news WHERE id in(58)

24_22 fieldExists 检测表字段是否存在

检测 news 表是否存在 title 字段

$db->fieldExists('title','news');

24_23 tableExists 检测表是否存在

M()->tableExists('category');

24_24 getVersion 获得数据库版本信息

$db->getVersion();

24_25 getLastSql 获得最后一条 SQL

$db->getLastSql();

24_26 getAllSql 获得所有 SQL 语句

$db->getAllSql();

24_27 getAffectedRows 获得受影响的行数

$db->getAffectedRows()

24_28 getInsertId 获得最后插入的主键值

```
1 $data=array('title'=>' 后盾 HDPHP 框架 ');
2 $db->insert($data);
3 $db->getInsertId();
```

24_29 getAllTableInfo 获所有表信息

获得当前数据库的所有表信息，数据大小包括碎片、数据、索引

```
M()->getAllTableInfo();
```

24_30 getDataBaseSize 获得数据库大小

获得当前数据库大小即所有表碎片、数据、索引之和

```
M()->getDataBaseSize();
```

24_31 getTableSize 获取表大小

获得 news 表大小，包含表碎片、数据、索引之和

```
M()->getTableSize('news');
```

24_32 createDatabase 创建数据库

以 gbk 编码创建数据库 hdphp

```
M()->createDatabase('hdphp','gbk');
```

24_33 truncate 清空表

清空表 news 表并将自增数归零

```
M()->truncate(news);
```

24_34 repair 修复表

```
M()->repair('user');
```

24_35 optimize 优化表

```
M()->optimize('user');
```

24_36 rename 修改表名

将 user 表更名为 hd_user 表

```
M()->rename('user','hd_user');
```

24_37 dropTable 删除表

```
$db->dropTable('hdphp');
```

24_38 beginTrans 开启事务

完成事务处理需要选择表引擎如 InnoDB、NDB、BDB

```
1 $data=array('wages'=>100);
2 $db->beginTrans();// 开启事务
3 $db->add($data);// 添加数据
4 $db->commit(); // 提交事务
```

24_39 rollback 事务回滚

当事务完整性被破坏或者其他原因可以通过 rollback 方法放弃本次事务操作

```
$db->rollback()
```

24_40 提交事务 commit()

如果整个事务完成正确可以通过 commit() 进行事务的提交完成最终操作

```
1 $data=array('wages'=>100);
```

```
2 $db->beginTrans();// 开启事务
3 $db->add($data);// 添加数据
4 $db->commit();// 提交本次事务
```

25. SQL 缓存

HDPHP 框架提供了高效简便的 SQL 查询缓存操作，只需要设置配置项就可以将全站缓存配置完成。

也可以使用 cache 函数对单条查询语句设置缓存。

全局缓存配置项

配置项	说明
CACHE_SELECT_TIME	SQL SELECT 查询缓存时间（设置 >=0 的数）0 为永久缓存，-1 为不缓存
CACHE_SELECT_LENGTH	SELECT 结果条数超过这个值将不进行缓存

25_1 cache 缓存函数

cache() 方法设置高于配置文件的设置，所以可以用配置文件设置全局的一个缓存时间，针对某条 SQL 可以使用 cache() 方法来单独操作。

查询结果缓存 30 秒

```
M('news')->cache(30)->select();
```

26. where 查询语言

HDPHP 框架通过 where 方法操作 SQL 条件，可以指定多种参数类型，非常自由。

注：数组为参数时会过滤非法字段

使用字符串做为条件

```
$db->where('id>1 and id<100')->all();
```

SELECT * FROM hd_news WHERE id>1 and id<100

使用数组做为查询条件

```
1 $where['title'] = ' 后盾 ';
2 $where['nid'] = '100';
3 $where['_logic'] = ' OR ';
4 $db->where($where)->all();
```

得到的查询条件是：title =' 后盾 ' OR nid ='100'

使用数组做为参数

```
1 $map[]='id>100  ';
2 $map[]='title like "% 后盾网 %"';
3 $map['_logic']='OR';
```

得到的查询条件是：id>100 OR title like "% 后盾网 % ，条件字符串后可以加 OR、AND、XOR 等，如果不加时使用 AND 做为连接

链式操作

```
$db->where("title like '% 后盾网 %'")->where("id>2")->all();
```

查询语句是：SELECT * FROM hd_content WHERE title like '% 后盾网 %' AND id>2

26_1 表达式查询

表达式	含义
EQ	等于（ = ）
NEQ	不等于（ <>）

GT	大于（>）
EGT	大于等于
LT	小于
ELT	小于等于
[NOT] LIKE	模糊查询
[NOT] IN	(不在) IN 查询
[NOT] BETWEEN	(不在) 区间查询
EXP	表达式查询

EQ: 等于 (=)

```
$map['id'] = array('eq', 1)
```

得到的查询条件是： id =1

NEQ: 等于 (<>)

```
$map['id'] = array('neq', 1)
```

得到的查询条件是： id <>1

GT: 等于 (>)

```
$map['id'] = array('gt', 1)
```

SELECT * FROM hd_news WHERE id >1

EGT: 等于 (>=)

```
$map['id'] = array('egt', 1)
```

得到的查询条件是： id >=1

LT: 等于 (<)

```
$map['id'] = array('lt', 1);
```

得到的查询条件是：id <=1

ELT: 等于 (<=)

```
$map['id'] = array('egt', 1)
```

得到的查询条件是：id <=1

[NOT]like: 同 SQL 中的 LIKE

$map['title'] = array('like' , '% 后盾网 %');

得到的查询条件是：title LIKE '% 后盾网 %'

$map['title'] = array('like' , array('% 后盾网 %',' 快学网 %'),'OR');

得到的查询条件是：title LIKE '% 后盾网 %' OR title LIKE ' 快学网 %'

[NOT]BETWEEN: 同 sql 的 [not]between

$map['nid'] = array('between' , '1,2');

或 $map['nid'] = array('between' , array(1,2));

得到的查询条件是：nid BETWEEN 1 AND 2

[NOT]IN: 同 sql 的 [not]in

$map['id'] = array('in' , array(1,2));

得到的查询条件是：id IN (1,2)

EXP: 表达式

$map['title'] = array('exp' , " like '% 后盾网 %'");

得到的查询条件是：title like '% 后盾网 %'

模糊查询

$map['_string'] = "uid>1 or username = 'houdunwang'";

得到的查询条件是：uid>1 or username = 'houdunwang'

请求字符串查询

$map['_query']="nid=1&title= 后盾网 &_logic=OR";

得到的查询条件是：nid ='1' OR title =' 后盾网 '

区间查询

$map['uid'] = array(array('gt', 3), array('lt', 5), 'AND');

得到的查询条件是：uid >3 AND uid <5 ，最后一个可以为 OR、AND、XOR 如果不写为 AND

27. 触发器 (trigger)

触发器是指在执行 CURD 时动态执行的方法，类似于 Mysql 中的触发器。可以使用相应触发器完成数据初始化，以及操作结果二次处理的目的。

说明

a. 触发器必须在扩展模型中定义

27_1 触发器方法

方法	执行时机
__init	模型实例化
__before_insert(&$data)	添加数据前
__after_insert(&$data)	添加数据后
__before_delete()	删除数据前
__after_delete(&$data)	删除数据后
__before_update(&$data)	更新数据后前
__after_update(&$data)	更新数据后
__before_select()	查询数据前
__after_select(&$data)	查询数据后

27_2 开关触发器 trigger

有时有些动作不希望激活触发器，这时可以使用 trigger 方法开启或关闭。

开启触发器

```
M('new')->trigger(true);
```

关闭触发器

```
M('new')->trigger(false);
```

28. ViewModel 视图模型

视图是虚表，是从一个或几个基本表（或视图）中导出的表，视图是原始数据库数据的一种变换，是查看表中数据的另外一种方式。可以将视图看成是一个移动的窗口，通过它可以看到感兴趣的数据。 视图是从一个或多个实际表中获得的。

为了演示视图的使用，创建如下表

用户表 user

id	自增主键 ID
username	用户名
password	用户密码

用户信息表 user_info

id	用户附加信息表自增主键 ID
email	用户邮箱
address	用户地址
qq	用户 QQ
user_id	用户 ID

新闻表 news

id	新闻自增主键 ID
title	新闻标题
content	新闻内容

user_id 发布新闻的用户 ID

28_1 扩展模型中定义视图规则

Model 目录中创建扩展模型文件 UserModel.class.php

```
1  class UserModel extends ViewModel
2  {
3      public $view = array(
4          'user' => array(
```

```
5               '_as' => 'member',
6               '_type' => 'INNER',
7               '_field' => 'uid,username'
8           ),
9           'user_info' => array(
10              '_type' => 'INNER',
11              '_on' => 'member.uid=user_info.user_id',
12              '_field' => 'email,address'
13          ),
14          'news' => array(
15              '_on' => 'member.uid=news.user_id',
16              '_field' => 'title'
17          )
18      );
19  }
```

参数说明

_as	表别名
_type	关联关系，INNER、LEFT、JOIN 三种
_on	关联条件
_field	查询字段

29. 多表关联

HDPHP 框架提供了多表关联支持，使开发者可以轻松应对各种数据表的关联操作。

29_1 关联类型

一对一关联

一对一关联是最简单的关联形式，比如用户表与用户信息表就是一对一关联。

一对多关系

一对多关系是最常见的一种表关系，比如文章表与栏目表。一个文章属于一个栏目，一个栏目可以有多个文章，栏目表为一，文章表为多。两张表之间是一种从属的关系，往往我们会把关联的键放在多的一方。

多对多关系

多对多关系无法区分主表与从表，比如说学生表与学生所学课程表之间就是一种多对多关系，因为一个学生可以学多个课程，一个课程也可以有多个学生来学习。对于多对多的情况，创建一张中间表来产生一对多的关系。

30. RelationModel 关联模型

多表关联使用 RelationModel 高级模型类来完成，扩展模型必须继承 RelationModel 模型。

为了关联模型的使用，创建如下表

会员表 user

uid	ID
username	用户名

用户信息表 user_info

address	地址
uid	用户 ID

角色表 role

rid	ID
rname	角色名

会员角色关联表 user_role

uid	会员 id
rid	角色 id

栏目表 category

cid	栏目 ID
cname	栏目名

文章表 news

id	新闻 ID
title	标题
uid	用户 ID
cid	栏目 ID

30_1 参数说明

参数说明	说明
type	HAS_ONE 一对一 HAS_MANY 一对多 BELONGS_TO 一对多 MANY_TO_MANY 多对多
parent_key	主表关联字段
foreign_key	从表关联字段
field	表字段

30_2 关联查询

HAS_ONE

每篇文章包含一个发布者，即为 HAS_ONE, 查询结果是一维数组。

```
1  class UserModel extends RelationModel
2  {
3      public $table = 'user';
4      public $relation = array(
5          'user_info' => array(// 关联表
6              'type' => HAS_ONE, // 包含一条主表记录
7              'foreign_key' => 'uid', //user_info 表字段
8              'parent_key' => 'uid', //user 表字段
9              'field' => array('address' => 'dizi'), // 关联表检索的字段
10         )
11     );
12 }
```

控制器

```
1  class IndexController extends Controller {
2    function index() {
```

```
3       $db = K('user');
4       $row = $db->all();
5    }
6 }
```

HAS_MANY

```
1 class UserModel extends RelationModel
2 {
3     public $table = 'user';
4     public $relation = array(
5         'news' => array(// 关联表
6             'type' => HAS_MANY, // 包含一条主表记录
7             'foreign_key' => 'uid', //user_info 表关联字段
8             'parent_key' => 'uid', //user 表字段
9         ),
10    );
11 }
```

BELONGS_TO

```
1 class NewsModel extends RelationModel
2 {
3     public $table = 'news';
4     public $relation = array(
5         'category' => array(// 关联表
6             'type' => BELONGS_TO, //category 被 news 关联
7             'foreign_key' => 'cid', // 从表字段 (category 表 )
8             'parent_key' => 'cid', // 主表字段 (news 表 )
9         )
10    );
```

```
11 }
```

MANY_TO_MANY

一般情况下将多对多通过中间表转化为多个一对多关系，中间表的关联字段必须与关联的两张表主键字段名相同。

```
1  class UserModel extends RelationModel
2  {
3      public $table = 'user';
4      public $relation = array(
5          'role' => array(// 关联表
6              'type' => MANY_TO_MANY, // 多对多关系
7              'relation_table' => 'user_role', // 中间表
8              'foreign_key' => 'rid',//role 表字段
9              'parent_key' => 'uid',//user 表字段
10         )
11     );
12 }
```

关联添加

通过 HDPHP 框架可以快速完成关联数据的添加操作，程序员所要做的工作就是配置好关联操作模型参数即可，可以说是一劳永逸的事情

HAS_ONE

模型

```
1  class UserModel extends RelationModel
2  {
3    public $table = 'user';
4    public $relation = array(
5      'user_info' => array( // 关联表
6          'type' => HAS_ONE, // 包含一条主表记录
```

```
7           'parent_key' => 'uid', //user 表主键
8           'foreign_key' => 'uid', //user_info 表关联字段
9       )
10  );
11 }
```

控制器

```
1  $db = K('User');
2  $data = array(
3           'username' => ' 后盾网 ',
4           'user_info' => array(
5           'email' => '2300071698@qq.com',
6           'address'=>' 北京市后盾网总部 '
7       ),
8  );
9  $db->insert($data);
```

HAS_MANY

设置与 HAS_ONE 相同，只需要修改 type 值

BELONGS_TO

设置与 HAS_ONE 相同，只需要修改 type 值

MANY_TO_MANY

设置与 HAS_ONE 相同，只需要修改 type 值

30_3 关联更新

HAS_ONE

从表可以没有主键字段

模型

```
class UserModel extends RelationModel
{
    public $table = 'user';
    public $relation = array(
        'user_info' => array(// 关联表
            'type' => HAS_ONE, // 包含一条主表记录
            'foreign_key' => 'uid', //user_info 表字段
            'parent_key' => 'uid', //user 表字段
        )
    );
}
```

控制器

```
$db = K('User');
 $data = array(
    'uid' => 1,
    'username' => ' 向军 ',
    'user_info' => array(
        'address' => ' 香港中环 '
    )
);
$db->save($data);
```

HAS_MANY

与 HAS_ONE 使用相同，从表必须有主键字段

BELONGS_TO

与 HAS_ONE 使用相同，从表必须有主键字段

MANY_TO_MANY

与 HAS_ONE 方法相同，从表必须有主键字段

30_4 关联删除

HAS_ONE

模型

```
class UserModel extends RelationModel
{
    public $table = 'user';
    public $relation = array(
        'user_info' => array(// 关联表
            'type' => HAS_ONE, // 包含一条主表记录
            'foreign_key' => 'uid', //user_info 表字段
            'parent_key' => 'uid', //user 表字段
        )
    );
}
```

控制器

```
$db = K('User');
$status = $db->del(1);
```

HAS_MANY

使用方法与 HAS_ONE 一样

BELONGS_TO

为了数据完整性，只删除主表数据，从表不删除

MANY_TO_MANY

使用方法与 HAS_ONE 一样

31. 模板引擎

31_1 基础知识

HDPHP 模版引擎分开了逻辑程序和外在的内容，提供了一种易于管理的方法。可以描述为应用程序员和美工扮演了不同的角色，因为在大多数情况下，他们不可能是同一个人。例如，你正在创建一个用于浏览新闻的网页，新闻标题，标签栏，作者和内容等都是内容要素，他们并不包含应该怎样去呈现。模板设计者们编辑模板，组合使用 html 标签和模板标签去格式化这些要素的输出 (html 表格，背景色，字体大小，样式表，等等)。有一天程序员想要改变文章检索的方式 (也就是程序逻辑的改变)。这个改变不影响模板设计者，内容仍将准确的输出到模板。同样的，哪天美工想要完全重做界面也不会影响到程序逻辑。因此，程序员可以改变逻辑而不需要重新构建模板，模板设计者可以改变模板而不影响到逻辑。

HDPHP 模版引擎是编译型模版引擎，模版文件只编译一次，以后程序会直接采用编译文件，效率非常高。

31_2 配置项

```
'TPL_PATH'          => 'View',     // 模板目录
'TPL_STYLE'         => '',         // 风格
'TPL_FIX'           => '.html',    // 模版文件扩展名
'TPL_TAGS'          => array(),    // 模板标签
'TPL_ERROR'         => 'error',    // 错误信息模板
'TPL_SUCCESS'       => 'success', // 正确信息模板
'TPL_ENGINE'        => 'HD',       // 模板引擎 HD,Smarty
'TPL_TAG_LEFT'      => '<',        // 左标签
'TPL_TAG_RIGHT'     => '>',        // 右标签
```

31_3 assign 向视图层分配内容

基本使用

```
1  $user = array(
```

```
2    array(  'name'=>' 李四 ', 'age'=>33),
3    array(  'name'=>' 后盾 ', 'age'=>6,)
4  );
5  $this->assign('username',$user);
6  $this->display('index');
```

以数组形式分配

```
1  $data=array('name'=>' 后盾网 ','url'=>'houdunwang.com');
2  $this->assign($data);
```

31_4 display 显示内容

语法：

display($tplFile = null, $cacheTime = -1, $cachePath = null, $contentType = "text/html", $show = true)

参数	说明
tplFile	模版文件，默认为当前控制器方法
cacheTime	缓存时间 , 默认为 TPL_CACHE_TIME 配置项值
cachePath	缓存路径，默认为 APP_CACHE_PATH 常量值
contentType	文件类型 (默认 text/html)
show	是否输出， false 返回字符串

模块 /View 目录 / 控制器 /web.html

```
$this->display('web');
```

template 目录中的 list.html

```
$this->display('template/list');
```

缓存 30 秒，设置缓存目录

```
$this->display('b.html',30,'Cache');
```

31_5 fetch() 获得模板解析数据

display() 用于直接显示内容到浏览器，而 fetch() 方法将返回内容

31_6 isCache() 缓存是否失效

isCache() 方法检测缓存是否失。

```
1 class indexController extends Controller{
2   function index(){
3     if(!$this->isCache()){ // 缓存失效时执行
4       echo ' 没有缓存 ';
5     }
6     $this->display('index',60); // 指定缓存时间为 60 秒
7   }
8 }
```

如果缓存文件失效则重新查找数据，视图缓存可加快大并发时的数据加载问题。

31_7 读取系统变量

{$hd.get.cid}	读取 $_GET 中的值
{$hd.post.cid}	读取 $_POST 中的值
{$hd.request.cid}	读取 $_REQUEST 中的值
{$hd.const.CONTROLLER}	读取系统中的常量
{$hd.session.cid}	读取 $_SESSION 中的值
{$hd.cookie.cid}	读取 $_COOKIE 中的值
{$hd.server.HTTP_HOST}	读取 $_SERVER 中的值
{$hd.config.db_host}	读取配置项值
{$hd.language.title}	读取语言包值

31_8 变量调节器

变量调节器是对变量作用的函数，系统函数与用户自定义函数都可做为变量调节器。

说明：

a. 默认变量是做为第一个参数传递的

b. 如果变量位置不是第一个参数使用 @@ 指定

使用任意 PHP 函数

在 HDPHP 框架中可以使用任意 PHP 函数做为变量调节器使用

```
{$data|strtoupper}
```

变量调节器链式操作

```
{$data|substr:2,8|strtolower}
```

日期 (变量值不是第一个位置)

```
{$feild.date|date:'Y-m-d',@@}
```

31_9 截取字符

截取 10 个字符标题，后面以 ... 结束

```
{$field.title|hd_substr:10,'...'}
```

31_10 日期输出

当前时间

```
{|date:'Y-m-d',@@}
```

使用系统 date 函数

```
{$date|date:'Y-m-d',@@}
```

输出“年 - 月 - 日”格式的时间

```
{$date|hd_date}
```

输出“年 - 月 - 日 时 : 分 : 表”格式的时间

```
{$date|hd_date:'y-m-d h:i:s'}
```

使用 date 函数输出

```
{$date|date:'y-m-d h:i:s',@@}
```

31_11 模板中使用函数

HDPHP 框架支持在模板视图中使用任意函数

注意 : 函数名前加 | , 如 {|substr:'houdunwang.com',0,2}

使用函数

{|U:'index','username= 向军 &sex= 男 '}

使用分配的变量

{|p:$data}

31_12 default 默认值

当变量为空时用默认值替代

{$houdunwang|default:' 后盾网 '}

32. 模版标签

32_1 jsconst 标签

将 PHP 常量创建 JavaScript 变量

<jsconst/>

32_2 Literal 不解析标签

literal 标签区域内的数据将被当作文本处理，此时模板将忽略其内部的所有字符信息。

```
1 <literal>
2   <list from="$data">
3     后盾网 人人做后盾
4   </list>
5 </literal>
```

32_3 foreach 标签

与 PHP 中的 foreach 使用方法一致

语法

<foreach from=' 变量 ' key=' 键名 ' value=' 键值 '>

  内容

</foreach>

基本使用

```
1 <foreach from='$user' key='$key' value='$value'>
2  {$value|strtoupper}
3 </foreach>
```

多重嵌套

```
1 <foreach from='$user' key='$key' value='$value'>
2   <foreach from='$value' key='$n'value='$m'>
```

```
3       {$m}
4    </foreach>
5 </foreach>
```

32_4 list 标签

语法

```
1 <list from=' 变量 ' name=' 值 ' row=' 显示行数 ' empty=' 为空时显示内容 '>
2    内容
3 </list>
```

基本使用

```
1 <list from='$data' name='d' row='10' start="0" empty=' 没有数据 '>
2      {$d.cname}
3 </list>
```

表示每次间隔 2 条数据输出

```
1 <list from='$row' name='n' step='2'>
2    {$n.title}
3 </list>
```

从第 2 条数据开始显示

```
1 <list from='$row' name='n' start='2'>
2    {$n.title}
3 </list>
```

与其他标签使用

```
1 <list from='$data' name='n'>
2      <if value='$hd.list.n.first'>
3          {$hd.list.n.index}: 这是第一条数据 <br/>
4      <elseif value='$hd.list.n.last'/>
```

```
5          {$hd.list.n.index}: 最后一条记录 <br/>
6      <else/>
7          {$hd.list.n.index}:{$n.title}<br/>
8      </if>
9  </list>
10 共有 :{$hd.list.n.total} 条
```

$hd.list.n.first 是否为第 1 条记录

$hd.list.n.last 是否为最后一条记录

$hd.list.n.total 总记录数

$hd.list.n.index 当前循环是第几条

其中 n 为 标签的 name 属性值

32_5 if 标签

语法

<if value=' 条件 '>

内容

</if>

基本使用

```
1  <if value="$webname eq 'houdunwang'">
2    后盾网
3  </if>
```

与 else 结合使用

```
1  <if value='$webname eq "houdunwang"'>
2    后盾网
3  <elseif value='$webname eq "baidu"'/>
4    百度
5  <else/>
6    其他网站
```

```
7 </if>
```

32_6 empty 标签

HDPHP 框架支持使用 empty 标签，与 PHP 中的 empty 标签意义一样

```
1 <empty value='$config.dbname'>
2     不存在值时显示的内容
3 <noempty/>
4     存在值时显示的内容
5 </empty>
```

32_7 php 标签

模版中直接使用 <?php ...?>

32_8 js 标签

引入 JavaScript 文件，路径中可使用所有 URL 常量

```
<js file='__CONTROLLER_VIEW__/Js/js.js'/>
```

32_9 css 标签

引入 CSS 文件 ，路径中可使用所有 URL 常量

```
<css file='__CONTROLLER_VIEW__/Css/css.css'/>
```

32_10 include 标签

加载模板文件使用

```
<include file=' 文件路径 '/>
```

示例

```
<include file='__PUBLIC__/head.html'/>
```

上面的标签会加载模板目录下 public 目录下的 head.html 文件

33. 自定义模版标签

框架提供了方便快速的标签定义，大大减少代码量，实现快速网站开发。

后盾 HDPHP 框架标签定义基于面向对象思想开发，设置自定义标签简单、快速，下面我们来学习掌握 HDPHP 框架自定义标签的使用方法。

33_1 说明

a. 在模块或应用的 Tag 目录下创建标签类文件

b. 标签类文件名必须以 Tag.class.php 结尾

c. 标签类必须以 Tag 结尾

d. 必须包含属性 Tag

e. 必须继承 Tag 父类

f. __init() 为构造函数

g. 标签函数必须以下划线开始

33_2 配置项

框架使用 implode 函数导入文件，不设置路径将导入模块或应用 Tag 目录下的文件。

```
'TPL_TAGS' =>array('HtmlTag')// 模块 Tag 目录下的 HtmlTag.class.php 文件
'TPL_TAGS' =>array('Admin.Tag.HtmlTag')// 导入 Admin/Tag/HtmlTag.class.php 文件
```

33_3 创建自定义标签内容

块标签定义

```
1  class ContentTag extends Tag
2  {
3      // 标签声明
4      public $Tag = array(
5              // 支持 4 导嵌套的块标签
6              'hdphp' => array('block' => 1, 'level' => 4),
```

```
7           );
8       /**
9        * 测试标签
10       * @param  Array        $attr           属性
11       * @param String        $content        块内容
12       * @param Object        $hd             视图对象
13       * @return string
14       */
15      function _hdphp($attr, $content, &$hd)
16      {
17          return ' 这是一个标签 ';
18      }
19  }
```

使用方法

```
1   <hdphp row='2'> ...</hdphp>
```

行标签定义

行标签定义文件如下

```
1   class ContentTag extends Tag
2   {
3       // 标签声明
4       public $Tag = array(
5               // 支持 4 导嵌套的块标签
6               'hdcms' => array('block' => 0, 'level' => 0),
7           );
8       /**
9        * 测试标签
10       * @param Array  $attr          属性
```

```
11      * @param String $content      块内容 ( 行标签此值无用 )
12      * @param Object $hd           视图对象
13      * @return string
14      */
15     function _hdcms($attr, $content, &$hd)
16     {
17         return ' 这是一个标签 ';
18     }
19 }
```

34. 自动处理

HDPHP 提供了自动验证、自动完成、自动填充等高效的数据处理机制。

说明：

a. create 模型方法执行自动过滤、自动验证、自动完成、自动填充

b. 一般不需要单独执行某个自动处理功能，使用 create() 模型方法调用即可

c. 如果每个方法单独执行时，$this->map() 必须放在最后执行

35. 自动验证

HDPHP 框架中通过简单的配置就可以数据的验证处理。

注：必须执行 create 或 validate 模型方法，才会执行自动验证

35_1 验证规则设置语法

array(' 字段名 ',' 验证方法 ',' 错误信息 ',' 验证条件 ',' 验证时机 ')

字段名

$_POST 中的字段

验证方法

如验证方法为 check_user，使用优先级如下

a. check_user 模型方法

b. check_user 函数

c. validate.class.php 类方法

错误信息

错误信息会记录到模型对象的 error 属性中

验证条件

值	说明
1	有表单时
2	必须验证
3	不为空

验证时机

值	说明
1	插入时
2	更新时
3	插入与更新

35_2 验证示例演示

通过模型的 validate() 方法验证

```
1 $db = M('user');
2 $db->validate = array(
3   array('username', 'nonull', ' 用户名不能为空 ',2,3)
4 );
5 if(!$db->create()){
6   $this->error($db->error);
7 }
```

扩展模型中的定义

```
1 class userModel extends Model {
2   public $validate = array(
3     array('username', 'nonull', ' 用户名不能为空 ', 2,3),
4   );
5 }
```

35_3 Validate 验证类

方法说明

参数	说明
nonull	不能为空
email	邮箱
http	网址
tel	固定电话
phone	手机
user	用户名长度如 :user:5,20
maxlen	最大长度如：maxlen:10

minlen	最小长度如：minlen:10
num	数字范围 如：num:20,60
regexp	正则如：regexp:/^\d{5,20}$/
confirm	两个字段值比对如：confirm:password2
china	验证中文
identity	身份证

35_4 验证函数传递参数

用户名长度大于 6 小于 20

```
array('username', 'user:6,20', ' 用户名必须大于 6 小于 20', 2,3)
```

年龄介于 20 ~ 60

```
array('age','num:20,60',' 年龄必须大于 20 小于 60',2,3)
```

正则

```
array('tel','regexp:/\d{11}/',' 手机号不正确 ',2,3)
```

密码比对

```
array('password','confirm:password2',' 两次密码不一致 ',2,3)
```

35_5 自定义验证

模型验证方法，优先级最高（建议使用）

```
1  class UserModel extends Model {
2      public $validate = array(
3          array('username','isadmin',' 不是管理员 ',2,3)
4      );
5      public function isadmin($name, $value, $msg, $arg) {
6          if ($value != 'admin') {
7              return $msg;
```

```
8         }
9         return true;
10    }
```

}

自定义验证函数

$name 为字段名 $value 为字段值 $msg 为错误信息 $arg 为参数

```
1  function isadmin($name, $value, $msg, $arg) {
2      if ($value != 'admin') {
3          return $msg;
4      }
5      return true;
6  }
```

36. 自动完成

自动完成是在模型层对数据进行自动处理的操作过程。

注：必须执行 create 或 auto 模型方法，才会执行自动完成

36_1 自动完成语法

array(' 表单字段名 ',' 处理方法 ',' 方法类型 ',' 验证条件 ',' 处理时机 ')

参数	说明
表单字段名	字段名
方法名	方法或函数
执行方式	method 模型方法 function 函数 string 具体值
处理条件	1 存在字段时 2 必须处理 3 值不为空时 4 值为空时
处理时机	1 插入时 2 更新时 3 插入与更新

使用示例

```
1 class UserModel extends Model
2 {
3   public $auto=array(
4     array('addtime','strtotime','function',2,3),//addtime 字段执行 strtotime 函数
5     array('click',100,'string',2,1)// 插入时设置 click 值为 100
6   );
7 }
```

37. 自动映射

将字段暴漏给用户会带来潜在的安全隐患，框架提供了字段映射用于提高系统安全性。

注：

a. 必须执行 create 或 map 模型方法，才会执行自动映射

b. 映射后不会更改 $_POST 值，更改模型 data 属性

映射实例

前台

```
1 <form method='post'>
2 用户名 :<input type='text' name='mingzi'/>
3 <input type='submit'/>
4 </form>
```

模型

```
1 class UserModel extends Model
2 {
3     public $map = array(
4             'mingzi' => 'username',// 将 $_POST['mingzi'] 映射为 username
5     );
6 }
```

38. Backup 数据备份

数据是网站的生命，Backup 可轻松完成数据备份与恢复操作。

注意

$_GET['bid'] 与 $_GET['status'] 为备份类专用

38_1 backup 备份数据库

语法：Backup::backup($args = array())

参数说明

参数说明	说明
dir	备份目录
size	分卷大小，默认 200kb
step_time	间隔时间，默认 500 毫秒
structure	备份表结构，默认 true
database	数据库，默认为当前数据库

实例操作

```
1  public function backup()
2      {
3          $result = Backup::backup(
4              array(
5                  'size'      => 200,// 分卷大小
6                  'dir'     => 'backup/' . date("Ymdhis") . '/',// 备份目录
7                  'step_time' => 1,// 备份时间间隔
8              )
9          );
10         if ($result === false) {
11             // 备份发生错误
12             $this->error(Backup::$error, __ROOT__);
```

```
        } else {
            if ($result['status'] == 'success') {
                // 备份完成
                $this->success($result['message'], __ROOT__);
            } else {
                // 备份过程中
                $this->success($result['message'], $result['url'], 0.2);
            }
        }
    }
```

38_2 recovery 数据库还原

语法：Backup::recovery($args)

参数说明	说明
backup_dir	还原目录

还原实例

```
public function recovery()
    {
        $dir    = 'backup/201411251025316';// 备份目录
        $result = Backup::recovery(array('dir' => $dir));
        if ($result === false) {// 还原发生错误
            $this->error(Backup::$error, __ROOT__);
        } else {
            if ($result['status'] == 'success') {// 还原完毕
                $this->success($result['message'], U('index'));
            } else {// 备份运行中 ...
                $this->success($result['message'], $result['url'], 0.2);
```

```
12              }
13          }
14      }
```

39. 验证码

没有设置的参数，将采用配置文件中相应值 。

默认使用配置项参数

```
1 $code = new code();
2 $code->show();
```

自定义验证码参数

```
1 $code = new code( 宽度 , 高度 , 背景颜色 , 文字颜色 , 验证码字数 , 文字大写 );
2 $code = new code(200,30,'#dcdcdc','#f00',8,22);
3 $code->show();
```

40. 分页处理类

没有设置的参数，将采用配置文件中相应值。

40_1 构造函数

new Page($total, $row = '', $pageRow = '', $desc = '',

$setSelfPage = '', $customUrl = '', $pageNumLabel = '{page}')

参数	说明
total	总数
row	每页显示文章数
pageRow	页码数量
desc	分页文字设置如：array('pre' => '上一页', 'next' => '下一页', 'first' => '首页', 'end' => '尾页', 'unit' => '条')
setSelfPage	当前页
customUrl	自定义 url
pageNumLabel	页码变量，默认为 {page}

属性说明（以下为部分属性，更多属性请查看 Page 类）

属性	状态	说明
$staticTotalPage	static	总数
$staticUrl	static	自定义 URL 地址
$fix	static	静态后缀如 .html
$pageNumLabel	static	替换标签默认为 {page}

根据配置项提取分页数据

```
1 $db = M('demo');
2 $total = $db->count();
3 $page = new page($total);// 传入总数
4 $row = $db->limit($page->limit())->all($page->limit());// 查询当前页的数据
5 echo $page->show(2);// 使用第 2 种栏目显示页码，提供 1~5 中样式
```

自定义分页字符

```
$page = new page(100,10,6,2,array('pre'=>' 上一层 ','next'=>' 下一层 '));
```

Url 自定义

在全站静态等场景，需要自定义分页 url 地址，Page 类可以轻松应付

```
$page = new Page(88,10,6,'','','http://localhost/{page}.html','{page}');
```

注：也可以实例化前定义 Page::$staticUrl 的值

41. 图像处理类

41_1 图像缩略图处理

没有设置的参数，将采用配置文件中相应值。

语法

public function thumb($img, $outFile = '', $thumbWidth = '', $thumbHeight = '', $thunbType = '')

参数说明

参数	说明
$img	原文件
$outFile	缩略图文件
$thumbWidth	缩略图宽度
$thumbHeight	分页文字设置如：array('pre' => '上一页', 'next' => '下一页', 'first' => '首页', 'end' => '尾页', 'unit' => '条')
$thunbType	缩略图处理类型： 1: 固定宽度 高度自增　　2: 固定高度 宽度自增 3: 固定宽度 高度裁切 4: 固定高度 宽度裁切　　5: 缩放最大边　　6: 缩略图尺寸不变，自动裁切图片

实例演示

```
1 $img = new image();
2 echo $img->thumb('r8.jpg','hdphp.jpg',200,200,2);
```

41_2 图像水印处理

没有设置的参数，将采用配置文件中相应值。

语法

public function water($img, $outImg = '', $pos = '', $waterImg = '', $pct = '', $text = "")

参数说明

参数	说明
$img	原文件
$outFile	缩略图文件
$pos	水印位置 1 为左上、2 为顶部居中、3 为顶部居右、 4 为左侧垂直居中、5 为正中心、6 为右侧垂直居中、 7 为左下、8 为下中、9 为下右
$waterImg	水印图像
$pct	透明度
$text	文字水印内容

原图加水印

```
1 $img = new image();
2 $img->water('houdunwang.jpg');
```

水印图储存

```
1 $img = new image();
2 $img->water('houdunwang.jpg','houdunwang_2.jpg',2);
```

42. 上传类

HDPHP 提供非常方便的上传处理机制，上传类会返回所有上传成功的文件列表。

语法

public function __construct($path = null, $ext = array(), $size = null)

参数说明

参数	说明
$path	储存目录
$ext	允许类型
$size	允许大小

全部文件上传

```
1 $upload = new upload();
2 $uplofiles =$upload->upload();
```

只处理 up 表单

```
1 $upload = new upload();
2 $uplofiles =$upload->upload('up');
```

上传成功后返回数组说明

属性	说明
path	上传成功的文件路径
basename	文件名
filename	主文件名（不带扩展名）
size	文件大小
ext	文件扩展名
uptime	上传时间
url	URI 地址

43. Data 数据处理类

Data 数据处理类主要是对数组等数据进行处理，如无限级分类操作、商品规格的迪卡尔乘积运算等。

43_1 Data::tree() 获得树状结构

语法

```
static public function tree($data, $title, $fieldPri = 'cid', $fieldPid = 'pid')
```

参数	说明
$data	数组
$title	字段名称
$fieldPri	主键 id
$fieldPid	父 id

43_2 Data::channelList 获得目录列表

```
static public function channelList($data, $pid = 0, $html = " ", $fieldPri = 'cid', $fieldPid = 'pid', $level = 1)
```

参数	说明
data	操作的数组
pid	父级栏目的 id 值
html	栏目名称前缀，用于在视图中显示层次感的栏目列表
fieldPri	唯一键名，如果是表则是表的主键
fieldPid	父 ID 键名
level	等级（不需要传参数，系统运行时使用）

43_3 Data::channelLevel 获得多级目录列表（多维数组）

```
static public function channelLevel($data, $pid = 0, $html = " ", $fieldPri = 'cid', $fieldPid = 'pid', $level = 1)
```

参数	说明
data	操作的数组
pid	父级栏目的 id 值
html	栏目名称前缀，用于在视图中显示层次感的栏目列表
fieldPri	唯一键名，如果是表则是表的主键
fieldPid	父 ID 键名
level	等级（不需要传参数，系统运行时使用）

43_4 Data::parentChannel() 获得所有父级栏目

static public function parentChannel($data, $sid, $fieldPri = 'cid', $fieldPid = 'pid')

参数	说明
data	操作的数组
sid	子栏目
fieldPri	唯一键名，如果是表则是表的主键
fieldPid	父 ID 键名

43_5 Data::isChild() 判断是否为子栏目

static function isChild($data, $sid, $pid, $fieldPri = 'cid', $fieldPid = 'pid')

参数	说明
data	操作的数组
sid	子栏目 id
pid	父栏目 id
fieldPri	唯一键名，如果是表则是表的主键
fieldPid	父 ID 键名

43_6 Data::hasChild() 是否有子栏目

static function hasChild($data, $cid, $fieldPid = 'pid')

参数	说明
data	操作的数组
cid	栏目 cid
fieldPid	父 ID 键名

43_7 Data::descarte() 迪卡尔乘积

迪卡尔乘积静态方法主要方便实现商品规格的表示形式

```
static function descarte($arr, $tmp = array())
```

44. 目录操作

44_1 Dir::getExt() 获得文件扩展名

static function getExt($file)

获得文件扩展名

Dir::getExt('houdunwang.com.php');

44_2 Dir::tree() 遍历目录内容

获得目录下的所有文件及目录内容，包含详细信息，支持递归遍历

static function tree($dirName, $exts = '', $son = 0, $list = array())

参数	说明
$dirName	遍历的目录名
$exts	显示文件的扩展名
$son	是否递归遍历，默认只遍历当前目录
$list	递归遍历时使用的参数（不用设置）

44_3 Dir::treeDir() 只显示目录树

获得指定目录下的所有目录详细信息，支持递归遍历，参数说明如下

static function treeDir($dirName, $pid = 0, $son=0,$dirs = array())

参数	说明
$dirName	遍历的目录名
$son	是否递归遍历，默认只遍历当前目录
$pid	父目录 ID
$dirs	递归遍历时使用的参数，调用本函数时不用设置

44_4 Dir::del() 删除目录或文件

static function del($file)

实例

```
1 Dir::del('hdphp');# 删除 hdphp 目录
2 Dir::del('houdunwang.php')# 删除 houdunwang.php 文件
```

44_5 Dir::create() 创建目录

本函数为创建目录使用支持递归创建，语法如下

static function create($dirName, $auth = 0777)

创建目录

```
dir::create('houdunwang/bbs');
```

44_6 Dir::copy() 复制目录内容

将目录内容包括子目录内容完整的复制到指定的目录中

```
static function copy($olddir, $newdir)
```

44_7 Dir::safeFile() 创建目录安全文件

Dir::safeFile() 在目录创建安全文件 index.html ，用于隐藏目录列表。

```
static function safeFile($dirName, $recursive=false)
```

参数	说明
$dirName	目录名
$recursive	是否在子目录也递归创建安全文件，默认不创建

45. string 字符串处理类

45_1 string::toSemiangle 全角转半角

将全角字符 【】转为半角字符 []

```
string::to_semiangle('【】');
```

45_2 string::pinyin 中文转拼音

获得“后盾网”中文的拼音

```
string::pinyin(' 后盾网 ');
```

注：如果转换的字符集与配置文件不同可以通过传递第 2 个参数指定字符集

45_3 String::removePunctuation() 去除标点符号

去除字符串中的所有半角或全角标点。

```
echo String::removePunctuation(' 北京后盾网，免费开源框架 hd。');
```

以上执行后将会把字符串 ' 北京后盾网，免费开源框架 hd。' 中的 '，。' 去除掉

45_4 utf8 转 gbk

```
String::utf8ToGbk(" 后盾网人人做后盾 ");
```

45_5 gbk 转 utf8

```
String::gbkToUtf8($data)
```

46. 缓存控制

为了在大并发时提供更快的响应速度，HDPHP 框架提供了缓存处理机制，操作简单、高效，涵盖 file 缓存、memcache 缓存、redis 可以根据需要指派不同的缓存处理机制，只需要在配置文件中设置即可。

46_1 影响缓存的配置项

对缓存影响的配置项

配置项	说明
CACHE_TYPE	缓存类型，可选择类型有：file[文件缓存] memcache[memcache 内存缓存]
CACHE_TIME	全局默认缓存时间 如果缓存时没有指定时间将以此为准，单位秒 ,0 为不缓存 -1 为永久缓存
CACHE_ZIP	缓存数据是否压缩 true 压缩 false 不压缩
SQL SELECT	查询缓存时间 推荐使用模板缓存 0 为关闭 -1 为永久缓存
CACHE_SELECT_TIME	SELECT 中的字段按需取不要取无用字段字段按需取不要取无用字段
CACHE_SELECT_LENGTH	SELECT 结果超过这个值不进行缓存
CACHE_TPL_TIME	模板缓存时间 0 为不缓存 -1 为永久缓存
CACHE_MEMCACHE	当缓存驱动为 Memcache 时设置 Memcache 主机
CACHE_REDIS	当缓存驱动为 Redis 时设置 Redis 主机
CACHE_SAVE	记录缓存命中率

46_2 Memcache 缓存设置

如果启用 Memcache 缓存控制，需要以下几项：

a. 设置配置项 CACHE_TYPE 为 memcache

b. 设置配置项 CACHE_MEMCACHE，各参数说明如下表

参数	默认值	说明
host	127.0.0.1	主机

port	11211	端口
timeout	1	连接超时时间（单位为秒，不要设置过长）
weight	1	权重（设置多个 memcache 服务器时有效，可以不用设置）
pconnect	1	是否持久连接（可以不用设置）

46_3 Redis 缓存设置

如果启用 Redis 缓存控制，需要以下几项：

a. 设置配置项 CACHE_TYPE 为 redis

b. 设置配置项 CACHE_REDIS，各参数说明如下表

参数	默认值	说明
host	127.0.0.1	主机
port	6379	端口
password	主机密码	没有密码时留空
timeout	1	连接超时时间（单位为秒，不要设置过长）
db	1	使用的数据库
pconnect	0	是否为长链接

46_4 CACHE 缓存函数

语法：cache($options)

不同的缓存方式所需要的参数是不同的

所有缓存方式公用配置项

配置项	说明
driver	缓存的类型
expire	默认的缓存时间
prefix	缓存前缀，区分不同控制器或方法中的同名 KEY
length	队列长度，队列即允许缓存的数量
zip	是否启动缓存数据压缩存储

不同缓存类型的配置项说明

配置项	配置项	说明
file 缓存机制	driver	file 缓存时必须设置为 file
	dir	缓存存放目录
memcache 缓存	driver	memcache 缓存时必须设置为 memcache
	host	array(// 多个服务器传入设置二维数组 'host'=>'127.0.0.1',//127.0.0.1 主机 'port'=>11211,//11211 端口 'timeout'=>1,// 连接超时时间 'weight'=>1,// 权重（多台服务器时有效设置） 'pconnect'=>1,// 是否持久连接（可以不用设置）)
Redis 缓存机制	driver	redis 缓存时必须设置为 redis
	host	array(// 多个服务器传入设置二维数组 'host'=>'127.0.0.1',// 主机地址 'port'=>6379,// 端口 'password'=>'',// 主机密码 没有密码时留空 'timeout'=>1,// 连接超时时间 单位为秒 'db'=>1,// 使用的数据库 'pconnect'=>0,// 是否为长链接)

方法说明

方法	说明
set($name,$value)	设置缓存数据
get($name)	获得缓存数据

del($name)	删除缓存
delAll($time=null)	删除所有缓存，如果设置 $time 则删除超过此值的缓存
options($name,$value=null)	设置与获得缓存配置

46_5 设置缓存

程序员根据网站要求指定合适的缓存时间，达到最佳缓存效果，操作缓存需要实例化缓存对象。

注：以下代码中的 Cache::init() 方法不传递参数时使用配置文件设置

示例 1

```
1 $cache  = cache::init();
2 $data = array(// 要缓存的数据
3         'webname' => ' 后盾网 ',
4        'time' => time()
5 );
6 $cache->set('houdunwang',$data,800);
```

如果没有指定缓存时间，将采用配置文件中 CACHE_TIME 选项的值

示例 2

```
1 $cache = cache::init(array('dir'=>'cache','zip'=>true,'prefix'=>'hd_'));
2 $cache->set('hd',' 向军 ');// 缓存数据 KEY 为 hd
3 echo $cache->get('hd');// 得到 KEY 为 hd 的缓存内容
```

将缓存数据放入根目录下的 cache 目录，以 hd_ 为缓存文件前缀，启用压缩数据

46_6 删除缓存

删除 hd 缓存数据

```
1 $cache = cache::init();
2 $cache->hd=null;// 删除缓存数据
```

使用函数删除 hd 缓存

```
$cache->del('hd');
```

46_7 删除所缓存

删除所有缓存

```
1 $cache = cache::init();
2 $cache->delAll();
```

删除 1 小时前的缓存文件

```
1 $cache->delAll(3600);
```

46_8 获得与设置配置

获得缓存配置

```
$cache = cache::init();
$cache->options('expire');
```

设置缓存参数

```
$cache->options('expire',600);
 echo $cache->options('expire');
```

46_9 缓存队列

为了避免缓存过多，可以通过 delAll() 删除过期缓存，也可以使用 HDPHP 框架中的缓存队列特性，即指定缓存的数量，以后进先出的原则删除最旧的缓存。

注：Memcache 等缓存控制本身具有 LRU 机制，所以缓存队列目前适用于 File 缓存

```
1 $cache = cache::init(array('length'=>2,'zip'=>false));
2 $cache->set('name',' 后盾网 ');
3 $cache->set('web','www.hodunwang.com');
4 $cache->set('bbs','bbs.hodunwang.com');
```

设置了 length 即队列长度为 2 所以只能缓存 2 条记录，队列采用先进先出的原则，所

以 name 将被删除

46_10 以 Memcache 操作缓存

无论是使用 Memcache 或者 Redis 操作缓存，可以通过修改配置文件的方式 ，或者调用 cache::init() 方法时传递参数即可，示例如下：

```
$cache = cache::init(array(
    'driver'=>'memcache',
    'host'=>array('host'=>'127.0.0.1','port'=>11211))
);
$cache->set('webname',' 后盾网 ');
echo $cache->get('webname');
```

46_11 以 Redis 操作缓存

无论是使用 Memcache 或者 Redis 操作缓存，可以通过修改配置文件的方式 ，或者调用 cache::init() 方法时传递参数即可，示例如下：

```
$cache = cache::init(array(
'driver'=>'redis',
    'host'=>array('host'=>'127.0.0.1','port'=>6379))
);
$cache->set('webname',' 后盾网 ');
echo $cache->get('webname');
```

46_12 S() 缓存函数

上面学习了通过 cache 类操作缓存，其实 S 方法也是通过 Cache 类操作缓存，只是提供了一种快捷的函数调用方式。

如果使用 memcache 或 redis 进行缓存操作，请修改配置文件中的相应配置项

S($name, $value = false, $expire = null, $options = array())

参数	说明

$name	缓存名称
$value	缓存内容
$expire	缓存时间
$options	缓存对象参数如 array('dir'=>'cache','driver'=>'memcache' , 'host'=>array('host'=>'127.0.0.1','port'=>11211,'timeout'=>1))

缓存数据

S('houdunwang',' 后盾网 ');

echo S('houdunwang');

删除缓存

S('houdunwang',null)

设置缓存驱动

S('houdunwang','bbs.houdunwang.com',1440,array('dir'=>'cache','driver'=>'file','zip'=>true));

将数据缓存到 cache 目录，以 file 文件形式缓存，启用数据压缩

46_13 F() 快速文件缓存

通过 F() 方法以文件形式快速对数据进行缓存

function F($name, $value = false, $path = CACHE_PATH)

参数	说明
$name	缓存名称
$value	缓存内容
$path	数据存放目录

缓存数据

F('houdunwang',array('name'=>' 后盾网 ','url'=>'houdunwang.com'));

获得数据

```
echo F('houdunwang');
```

删除缓存

```
F('houdunwang',NULL)
```

47. 多语言支持

后盾 HDPHP 框架提供便捷的多语言支持，程序开发人员只需要配置好语言包即可，下面来掌握具体的使用方法。

a. 首先创建语言包即数组文件，放置在应用组中的 Common/language 目录或应用的 Extend/language 目录中都可，其他如果两处都有语言包文件则应用中的语言包文件优先级更高

b. 使用语言包可以通过修改配置项 LANGUAGE 或在控制器中通过 C('LANGUAGE','zh') 来进行设置

语言包 zh.php 文件

```
1  <?php
2  if(!defined('HDPHP_PATH'))exit;
3  return array(
4    'title'=>' 后盾语言测试 '
5  );
6  ?>
```

控制器内容

```
1   class indexController extends Controller{
2    function index(){
3        C('language','zh');// 也可以修改配置文件中 language 值
4    }
5  }
```

在视图中可以通过 {$hd.language.hd} 读取数据即可

48. 生成 HTML 静态文件

生成静态 html 好处很明显，第一点是非常有利于 SEO 的优化，搜索引擎可以对 html 文件进行更好的收录。第二点是显著减轻网站的负载，WEB 服务器在处理 html 文件时只是简单的读取操作，不会经由 PHP 模块进行处理，也不会有数据库服务器的操作，所以速度要较 PHP 文件快得多。

HTML 有很多特点，但是我们也不能盲目使用 HTML，而是应该在更新频率较低的页面采用 HTML 处理方式 ，比如说文章系统、博客系统，社区门户等方面，而在更新频率较高的应用如论坛、微薄、SNS 等方面建议采用 HDPHP 框架的 PATHINFO 及结合 HDPHP 框架缓存机制，解决 SEO 及高负载问题，所以请根据情况适时的选择合适的处理机制。

后盾 HDPHP 框架提供高效简单的批量文件处理功能，我们来通过示例来解释 HDPHP 框架生成 HTML 文件的使用方法。

48_1 createHtml 生成静态文件

function createHtml($htmlFile, $htmlPath, $template)

参数说明

配置项	说明
$html	文件名
$htmlPath	存放目录
$template	模板文件

49. cart 购物车类

HDPHP 框架为了便于商城系统的开发提供了完善的购物车处理类，使商城购物车处理更加方便快捷，程序员只需要专注业务流程而不用关注实现步骤，大大增加了开发效率影响购物车类的配置项

配置项

配置项	说明
CART_NAME	$_SESSION 中购物车的名称

静态类 Cart 属性说明

属性	说明
Cart::$cartName	储存在 $_SESSION 中的购物车名称

49_1 添加购物车 cart::add()

添加购物车使用 cart::add() 方法实现，购物车中的数据会写入到 $_SESSION 超全局数组中，具体使用方法如下：

```
1  class indexController extends Controller{
2    function index() {
3        $data = array(
4            'id' => 1, // 商品 ID
5            'name'=>' 后盾网 PHP 视频教程光盘 ',// 商品名称
6            'num' => 2, // 商品数量
7            'price' => 988, // 商品价格
8            'options' => array(// 其他参数如价格、颜色、可以为数组或字符串
9            'color' => 'red',
10            'size' => 'L'
11       )
12   );
```

```
13  cart::add($data); // 添加到购物车
14  p($_SESSION);
15  }
16 }
```

49_2 更新购物车 Cart::update()

更新购物车需要传入商品的唯一 SID ,切记不是商品的 ID 值 ,同时传入更新的商品数量，

示例如下

```
1  $data=array(
2    'sid'=>'4d854bc6',// 唯一 sid，添加购物车时自动生成
3    'num'=>88
4  );
5  Cart::update($data);
```

49_3 获得购物车商品数据 Cart::getGoods()

通过 cart::getGoods() 可以获得购物车中的所有商品数据，使用方法如下

```
cart::getGoods();
```

49_4 获得购物车所有数据包 Cart::getAllData()

cart::getAllData() 与 cart::getGoods() 区别在于不仅获得所有商品数量，同时还会获

得购物车中的所有商品数量及总价格

```
cart::getAllData();
```

49_5 清空购物车中的所有商品 Cart::delAll()

清空购物车中的所有商品

```
cart::delAll();
```

49_6 获得商品总价格 Cart::getTotalPrice()

通过 cart::getTotalPrice() 方法可以得到购物车中的商品合计总价，使用方法如下：

```
cart::getTotalPrice();
```

49_7 统计购物车中的商品数量 Cart::getTotalNums()

通过 cart::getTotalNums() 方法得到购物车中的所有商品数量，使用方法如下：

```
cart::getTotalNums();
```

49_8 获得定单号 Cart::getOrderId()

通过 Cart::getOrderId() 方法用于获得商品唯一定单号

```
Cart::getOrderId();
```

50. RBAC 基于角色的权限控制

后盾 HDPHP 框架支持完善的，强大的基于角色的权限控制，所有操作都通过核心类 Rbac.class.php 完成

50_1 影响 RBAC 的配置项

配置项	说明
RBAC_TYPE	1 时时认证 2 登录认证
RBAC_SUPER_ADMIN	超级管理员 认证 SESSION 键名
RBAC_USERNAME_FIELD	用户表中的用户名字段名称
RBAC_PASSWORD_FIELD	用户表中的密码字段名称
RBAC_AUTH_KEY	用户登录后在 SESSION 中储存的用户 ID
RBAC_NO_AUTH	不需要验证的控制器或方法如：array(index/index) 表示 index 控制器的 index 方法不需要验证
RBAC_USER_TABLE	用户信息表
RBAC_ROLE_TABLE	角色信息表
RBAC_NODE_TABLE	节点信息表
RBAC_ROLE_USER_TABLE	角色与用户中间关联表
RBAC_ACCESS_TABLE	权限分配表

50_2 Rbac::IsLogin()

这个方法是验证用户是否登录，主要是检查 $_SESSION 值，因为用户登录后会在 SESSION 保存配置项中的 RBAC_AUTH_KEY 值，如果用户成功登录这个值是用户的表中的 ID 号 。

```
1 function index(){
2   if(Rbac::isLogin()){// 验证用户是否登录
3       echo ' 已经登录 ';
4   }else{
5       echo ' 没有登录 ';
6   }
7 }
```

如果用户没有登录则跳转到登录界面

50_3 Rbac::login() 用户登录

登录函数，登录成功将合法用户权限信息写入 $_SESSION，如果用户表中存在与参数 $superadmin 相同的用户则此用户为超级管理员 在 $_SESSION 中记录，超级管理员不受任何访问限制，语法如下

```
function Rbac::login($username, $password, $superadmin = null,
$fieldUserName = null,  $fieldPassword = null)
```

参数说明

a. $username 参数为用户名，用户错误可以通过 Rbac::$error 得到错误内容

b. $password 参数为密码，密码错误也可以通过 Rbac::$error 得到错误内容

c. $superadmin 参数为超级管理员的帐号名，也就是说如果这个用户的帐号与这个参数相同，并且登录成功，那么这个用户为超级管理员，超级管理员用户在网站中可以随意畅行，没有限制。

d. $fieldUserName 用户表中的用户名字段名称

e. $fieldPassword 用户表中的密码字段名称

如果发生错误，均可以通过 $this->error 得到错误信息内容

```
1 if(!Rbac::login($_POST['username'],$_POST['password'],'admin')){
2   $this->error(Rbac::$error,'index/login');
3 }
```

以上代码是用户登录验证，如果用户名正确且密码正确，用户成功登录，同时将用户权限信息写入 SESSION，如果登录失败则显示失败原因，同时跳转到登录窗口界面。

50_4 Rbac::checkAccess()

这个方法主要是验证用户是否有访问应用、控制器模块、方法的权限，如果没有将返回 FALSE 值，程序员可以根据返回值来决定操作方向。

```
1 if(!Rbac::checkAccess()){
2     $this->error(' 对不起你没有操作权限 ');
3 }
```

如果一个控制器中的所有方法都要验证，可以定义控制器中的 __auto 方法，这个方法会自动执行，也就是说执行一个控制器方法会首先运行 __auto 方法

也可以定义一个公共的 RbacController 类来完成验证操作，将所有有关验证的代码全部放在 rbacController 里的 __auto 方法中，控制器需要验证时只需要继承 RbacController 控制器即可

50_5 Rbac::getNodeList() 获得所有节点列表

通过本函数可以获得数据表中的所有节点列表信息，用于在后台构造权限控制视图时使用，就像 HDPHP 框架中的 SETUP 应用中的 RBAC 设置

function getNodeList($roleId = '')

$roleId　　角色 id 如果传值将获得角色的所有权限信息

示例：

```
1 class indexController extends Controller{
2     function index(){
3         $list = Rbac::getNodeList();
4         p($list);
5     }
6 }
```

51. session 操作

HDPHP 框架提供内置的 SESSION 处理引擎如 Mysql 引擎、memcached 引擎、redis 引擎等，同时提供了便捷的 session 操作方式，使操作 session 更加安全高效。

如果使用 HDPHP 框架 session 类操作 session 时程序员不用执行 session_start() 方法，系统会自动进行判断自动开启 session 功能。

SESSION 的操作可以方便的通过 session() 函数

语法：session($name,$value='',$options)

功能	代码
设置 session	session('webname',' 后盾网 ')
获得 session	session('webname');
删除 session	session('webname',null);
检测 session	session('?webname');
注销 session	session('[destroy]')
停止 session	session('[pause]');
开启 session	session('[start']);

修改 session_name

建议通过修改配置项 'SESSION_NAME' 改变 session_name 提高网站安全性。

52. 自定义 session 处理机制

HDPHP 框架提供了多种 session 处理机制包括 Mysql、memcache、redis 等

影响 session 的配置项

配置项	说明
SESSION_NAME	储存 SESSION_ID 的 COOKIE 名称
SESSION_TYPE	支持 mysql,memcache,redis 等 session 处理机制
SESSION_OPTIONS	SESSION 处理参数

52_1 session 的 mysql 处理机制

系统提供了基于数据库的 session 处理机制，使 session 处理更加高效，同时也便于在线人数统计等操作的实现，修改配置项 SESSION_OPTIONS 即可。

注：除了表名外，其余配置项可以不设置

```
'SESSION_TYPE'=>'mysql',// 必须留空
'SESSION_OPTIONS'=>array(
  'host'     =>'127.0.0.1',//Mysql 主机
  'port'     =>3306,// 端口
  'user'     =>'root',// 用户名
  'password'  =>'',// 密码
  'database'  =>'test',// 数据库
  'table'        =>'hd_session',// 完整表名
  'expire'   =>1440,//session 过期时间
)
```

创建数据表

SESSION 表结构如下

```
create table hd_session(
  sessid char(32) primary key not null,#session_id
```

```
3    card char(32) not null,#SESSION 令牌
4    data text not null,
5    atime int(10) not null,
6    ip char(15) not null
7  )charset utf8 engine=myisam;
```

52_2 session 的 Memcache 处理机制

HDPHP 框架支持 Memcache 的 session 操作机制，速度更快，在大并发，大请求发生时效果尤为明显，使用 HDPHP 框架的 Memcache 操作 session 非常简单，需要修改配

置项如下：

```
1  'SESSION_TYPE'=>'memcache',// 必须留空
2  'SESSION_OPTIONS'=>array(
3    'host'=>'127.0.0.1',// 主机
4    'port'=>11211// 端口
5  )
```

52_3 session 的 Redis 处理机制

因为 Redis 数据操作都是在内存中所以在大并发下也会带来惊人的速度提升。

配置项设置

```
1  'SESSION_TYPE'=>'redis',// 必须留空
2  'SESSION_OPTIONS'=>array(
3    'host'=>'127.0.0.1',// 主机
4    'port'=>6379,// 端口
5    'password'=>'',// 密码
6    'db'=>0,// 数据库
7  )
```

53. cookie 操作

通过 cookie() 函数可以任意且方便对 cookie 操作，具体使用方法如下：

cookie 的使用可以方便的通过 cookie() 函数完成

语法：cookie($name,$value='',$options=null)

功能	代码
设置 cookie	cookie('webname','houdunwang',100);//100 为过期时间，为 0 表示会话时间
获得 cookie	cookie('webname')
删除 cookie	cookie('webname',null)
删除指定前缀所有 cookie	cookie(NULL,'hd_')

参数 options 说明

名称	说明
prefix	cookie 前缀（默认为配置文件设置）
expire	默认过期时间，0 为会话时长（默认为配置文件设置）
path	保存路径 （默认为配置文件设置）
domain	有效域名（默认为配置文件设置）

54. Xml 操作类

在 web 开发中经常大量用到 XML 这种快速简便的数据传输格式，HDPHP 框架为程序员提供了简单快速的 xml 处理机制

54_1 Xml::create() 创建 xml 文件

语法：xmlCreate($data, $root = null, $encoding = 'UTF-8')

$data	数据
$root	根节点
encoding	编码方式默认为 utf-8

示例：

```
1  class indexController{
2    function index() {
3      $data = array(
4        'news' => array(
5         'title' => ' 后盾网 HDPHP 框架 ',
6        'con' => 'HDPHP 框架为 WEB 开发助力 '
7        )
8      );
9    header('content-type:text/xml');
10   echo xml::Create($data, 'root', 'utf-8');
11   exit;
12   }
13 }
```

54_2 Xml::toArray()

将 XML 文件转为标准的数组结构，本函数需要参数为 xml 内容的字符串数据

示例

```
1  class indexController {
2   function index() {
3       p(xml::toArray(file_get_contents('xml.xml')));
4   }
5  }
```

55. 错误异常处理

后盾 HDPHP 框架错误处理非常高效全面，可以针对需要自定显示及错误处理方式。

55_1 开启调试模式

在单入口文件定义常量 define('DEBUG',true)

55_2 自定义错误内容显示方式

HDPHP 框架可以完全根据程序员要求，自定义显示错误内容，因为全面的错误提 示显示的内容会比较多，有时我们不需要得到这么多信息，比如说在 AJAX 测试时返回太多 信息不便于代码纠错 ,那么我们可以个性配置文件中的相应配置项轻松实现自定义的显示内容。

在后盾 HDPHP 框架产品中还有一项人性化的设置，当系统关闭调试模式后，为了避免出现白页的情况，可以在配置文件中指定配置项来显示友好的文字内容。

有关错误处理的配置项

配置项	说明
404_URL	404 跳转 url
ERROR_URL	错误跳转 URL
ERROR_MESSAGE	关闭 DEBUG 显示的错误信息
SHOW_NOTICE	是否显示 Warning 与 Notice 错误显示

56. 日志处理

HDPHP 框架的日志功能非常完善，根据不同错误类型提供完善的处理机制，在关闭调试模式后，也可以查看系统日志来对系统运行情况进行全面了解。日志文件会完整的保存出错文件名、代码行号、函数名、出错时间等详细信息。

配置项

配置项	说明
LOG_SIZE	日志文件大小
LOG_RECORD	记录日志
LOG_LEVEL	写入日志的错误级别
LOG_EXCEPTION_RECORD	记录异常

57. 函数

57_1 C 设置配置项

C() 函数可以方便的读取与设置配置项。

示例

```
1 class indexController extends Controller {
2   function index() {
3       echo C('db_host');// 输出配置文件中的 db_host 配置项
4        echo '<br/>';
5       echo C('db_host','localhost');// 设置配置项 db_host 为 localhost
6   }
7 }
```

输出结果为 127.0.0.1 与 localhost

57_2 O 函数

O() 函数用于生成对象，或者生成对象后执行类方法同时可以传入参数

参数

$class	类名称
$method	方法名称
$args	参数（数组形式）

示例

```
1 class indexController extends Controller {
2   function index() {
3       O('indexController','html',array(2,3));
4   }
5 }
```

执行 indexController 中的 html 方法并且传入参数 2,3

57_3 controller 实现化控制器

controller() 方法可以方便的实例化控制器对象

实例化当前应用中的 UserControl 控制器

```
Controller('User');
```

实例化 Admin 应用的 LoginControl 控制器

```
controller('Admin/Login')
```

57_4 tag 调用标签

tag() 函数可以方便在的在控制器或模板视图中调用标签

控制器中调用标签

```
1  class IndexController extends Controller
2  {
3    protected $upload;
4    public function index()
5    {
6      $up = tag('upload',array('name' => 'upfile','limit'=>1, 'width' => 88, 'height' => 78));
7      $this->assign('up', $up);
8      $this->display();
9      }
10   }
```

模板视图中调用标签

```
{|tag('upload',array('name' => 'upfile','limit'=>1, 'width' => 88, 'height' => 78))}
```

57_5 load 文件加载

load 提供方便的文件加载处理

加载应用 Extend/Lib 目录下的 functions.php 文件

load('functions');

加载应用目录下 Common 目录下的 functions.php 文件

load('@.Common.functions');

加载网站根目录下 Common 目录下的 functions.php 文件

load('./Common/functions');

57_6 404 错误处理

处理 404 错误，开启 debug 时会抛出异常，关闭后会加载 404 错误模板文件。

_404(__METHOD__.' 页面没找到 ','404.html');

57_7 addslashes_d 转义字符串

系统提供了转义函数 addslashes 但只能转义字符串，通过本函数不仅可以对字符串进行转义也可以对数组及对象中的属性值的内容进行转义。

```
1 $data = array(
2   'username'=>' 向 ' 军 ',
3   'msg'=>'' 后盾网 '\PHP\ 教育 '
4 );
5 $data =  addslashes_d($data);
```

57_8 array_change_key_case_d 改变数组键名大小写

PHP 提供了一个将数组的键名转为大写或小写的函数，但是只能转换一维数组，本函数可以方便的将多维数组的键名转为大写或者小写，参数 1 为大写，0 为小写。

```
1 $arr = array(
2   'name'=>' 后盾网 ',
```

```
3       'url'=>array(
4           'bbs'=>'bbs.houdunwang.com',
5           'edu'=>'edu.houdunwang.com'
6       )
7   );
$newArr = array_change_key_case_d($arr);
```

57_9 array_change_value_case 数组的值大小写转换

本函数用于转换数组值的大小写，支持递归的值大小写转换

57_10 array_defined 将数组转为常量

传入数组将数组的键名做为常量名，键值做为常量值

```
array_defined(array('WEBNAME'=>' 后盾网 '));
```

57_11 array_key_exists_d 不区分大小写检测键名是否存

在 array_key_exists_d() 提供与系统函数 array_key_exists() 同样的功能，只是不区分大小写 。

```
var_dump(array_key_exists_d('Hd',array('hd'=>' 后盾网 ')));// 返回真
```

57_12 array_to_String 将数组转为字符串表示形式

57_13 controller 实例化控制器对象

语法：function controller($controller)

实例化当前应用中的 index 控制器对象

```
controller('index')
```

实例化 member 应用 index 控制器对象

```
controller('member/index')
```

57_14 A 执行控制器中的方法（支持分组）

语法：function A($arg, $args = array())

执行动作并传参

A('show',array('hdphp',2030));

执行当前控制器中的 show 方法，并参数参数 hdphp 与 2030

执行控制器动作

A('User/add');

执行 User 控制器中的 add 方法

指定模块的动作

controller('member/index/show');

执行 member 应用中的 indexControl 控制器的 show 方法

57_15 encrypt 加密处理

encrypt($data, $key = null)

$data　　加密字符串

$key　　密钥

加密数据

encrypt(' 这是加密数据 ');

57_16 decrypt 解密方法

解密操作

decrypt(encrypt(' 后盾网人人做后盾 '));

57_17 p 打印函数

由于 print_r 等打印函数会将内容在一行输出不便于阅读，通过本函数可以分行格式化输出内容，更加便于阅读加快调试速度。

57_18 halt 输出错误信息

如果开启 DEBUG 时输出错误信息内容，当 DEBUG 关闭时调用 _404() 函数输入 404 页面。无论 DEBUG 开启与关闭都会终止 PHP 脚本执行

57_19 firephp 调试插件

需要 firefox 下安装 firebug 和 firephp 插件，使用 firephp() 方法进行调试非常方便，尤其是在查看 Ajax 数据时。

使用方法：http://bbs.houdunwang.com/thread-53710-1-1.html

57_20 extension_existsPHP 扩展模块是否存在

通过本函数判断 php 扩展模块是否存在。

判断 mysqli 扩展是否在存在

```
extension_exists('mysqli);
```

57_21 file_exists_case 区分大小写的文件判断

```
file_exists_case('index.php');
```

57_22 int_to_string 数组进行整数映射转换

```
int_to_string(array(array('status'=>1)),array('status'=>array('0'=>' 关闭','1'=>' 开启 '));
```

57_23 get_defines 获得常量

获得常量系统 ，默认为获得用户定义常量

```
get_defines($name = '',$value=null, $type = 'user')
```

获得所有用户定义常量

```
get_defines();
```

获得常量名为 APP 的值

```
get_defines('APP');
```

HD 常量存在返回其值，否则返回 ' 后盾 '

```
get_defines('HD',' 后盾 ');
```

获得所有系统常量

```
get_defines('',true);
```

57_24 get_size 根据大小返回标准单位 KB MB GB 等

传入大小单位为字节，返回以 KB 或者 MB 或者 GB 为单位的字符串，比如说我们想在视图层显示文件大小，以 PHP 默认的字节单位不便于阅读而 get_size() 函数可以帮上你的忙。

```
get_size('1000');
```

57_25 go 跳转到指定 url

跳转到 http://bbs.houdunwang.com

```
go('http://bbs.houdunwang.com');
```

57_26 browser_info 获得浏览器版本

使用 browser_info() 可以方便的获得客户端浏览器，如果是 IE 浏览器将返回 msie9 等形式，开发者可以借此做出针对不同浏览器的业务处理。

57_27 image_type_to_extension 安全的获得图像扩展名

57_28 ip_get_client 显示客户端

57_29 is_ssl 是否为 SSL 协议

57_30 load 载入文件

load() 方法载入文件要比 include、include_once、require、require_once 更加高效，由于函数内置静态缓存机制，如果一个文件被多次加载时不会重复加载。

加载文件 config.inc.php

load('config.inc.php');

加载 应用目录 /Extend/Lib/functions.php 文件

load('@.Extend.Lib.funtions');

57_31 md5_d 方法

用于将任何内容包括数组、对象、字符串、数值类型生成 md5 序列化的字符串

```
1 $array=array('webname'=>' 后盾网 ');
2  echo md5_d($array);
3 # 输出结果 3deca1aa87fc07e0a649432408bdd4b2
```

57_32 mobile_area 电话号码来源

通过本函数获得电话号码所在地，在分类信息等网站来说是一个很实用的功能。

mobile_area('13121111111');

57_33 php_merge 合并 php 文件

多个 PHP 文件进行合并，如果将参数 $delSpace 设置为 1 则合并时会删除文件中的空格与注释

57_34 print_const 打印所有常量

由于 HDPHP 框架运行中，包含大量框架定义常量及用户代码定义常量，通过本函数可以方便的知道都有哪些常量定义了及其具体的内容值。

打印常量

print_const();

返回所有用户定义常量

print_const(false);

57_35 rand_str 获得随机字符串

57_36 set_http_state 设置 HTTP 状态信息

可传参数为：200,301,302,400,403,404,500,503

57_37 compress 压缩 PHP 代码

去空格，去除注释包括单行及多行注释，参数为字符串内容

compress('hdphp.php');

57_38 data_format 数据安全处理

将函数作用于数据，比如说可以在插入数据时对数据一次性进行转义或实体化处理，如果不存递第 2 个参数系统会使用

示例：

```
1 $_POST=array(
2   'html'=>'<script>alert(2);</script>',
3   'name'=>'houdunwang.com'
4 );
5 p(data_format($_POST,array(htmlspecialchars,strtoupper));
```

57_39 stripslashes_d 去除转义字符

去除数组或字符串中的转义内容，使用方法与 stripslashes_d 一样，就不举示例了。

57_40 throw_exception 抛出异常

57_41 remove_url_param 移除 URL 中 GET 参数

通过本函数可以移除 URL 中的指定 GET 变量，在进行参数项检索时很有帮助，比如分类信息应用中租房列表中的搜索会按价格、厅室、出租类型检索通过本函数可以为完成此功

能起到一定帮助

移除当前网址参数

当前 URL 为 http://localhost/index.php/index/city/beijing/price/300_500

url_param_remove('price');

移除指定 URL 参数

```
1 $url = 'http://localhost/index.php/index/city/beijing/price/300_500';
2 remove_url_param('price',$url);
```

57_42 date_before 获得几秒、几分、几年前

在微博，论坛等项目中，经常要获得微博发表于几分前，几年前等字符串表示，通过 date_before() 函数可以轻松实现，本函数需要 2 个参数，说明如下 :

参数说明

参数 1 : 要计算的时间戳

参数 2 : 时间单位，默认为 array(' 年 ',' 月 ',' 日 ',' 星期 ',' 小时 ',' 分钟 ',' 秒 ')

57_43 get_uuid 获得唯一值 uuid

UUID 是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的

获得唯一 uuid 以中线连接

get_uuid('-')