PHP官方手册（中文）：

https://www.php.net/manual/zh/

基本语法：

==========

echo是输出语句，用于输出字符串

PHP标记四种语法风格：

1、标准标记

以 <?php 开始，以 ?> 结束

这是最常用的标记类型，服务器不能禁用这种风格的标记。它可以达到更好的兼容性、可移植性、可复用性，所以PHP推荐使用这种标记。

2、短标记

以 <? 开始，以 ?> 结束

使用短标记，必须在配置文件php.ini中启用short_open_tag选项。另外，因为这种标记在许多环境的默认设置中是不支持的，所以PHP不推荐使用这种标记。

3、ASP标记

以 <% 开始，以 %> 结束

须在配置文件php.ini中启用asp_tags选项，在许多环境的默认设置中是不支持的，因此在PHP中不推荐使用这种标记

4、SCRIPT标记

以 <script language=”php”> 开始，以 </script> 结束

PHP一般不推荐使用这种标记，只需了解即可。

PHP注释

单行注释 / 或 #

多行注释 /* */

PHP变量

定义变量：$变量名

PHP的数据类型

gettype(传入一个变量) 能够获得变量的类型

标量类型： boolean（布尔型）Integer（整型）float（浮点型）string（字符串型）

复合类型： array（数组）object（对象）

resource（资源）

NULL（空值）

1、boolean布尔类型

$bool1 = true; //把true值赋给变量$bool1

2、integer整型

前面可加上“+”或“-”号表示正数或负数。 $b = -123; //十进制负数，数值-123

当使用八进制表示时，数字前必须加上0（零），$c = 0123; //八进制数，等于十进制的83

使用十六进制表示时，数字前必须加上0x，$d = 0x123; //十六进制数，等于十进制的291

3、float浮点型

浮点型可以存储整数，也可以存储小数

4、string 字符串

$a='字符串';

$b="字符串"; echo "
"; //会被解析成换行

包含在双引号的字符串会被解析，而包含在单引号中的字符串不会解析，只会输出其字符本身。

PHP中检测数据类型的相关函数

is_bool	检测变量是否属于布尔类型
is_string	检测变量是否属于字符串类型
is_float	检测变量是否属于浮点类型
is_integer	检测变量是否属于整型
is_null	检测变量是否属于空值
is_array	检测变量是否属于数组
is_resource	检测变量是否属于资源
is_object	检测变量是否属于对象类型
is_numeric	检测变量是否属于数字或数字组成的字符串

可变变量：将变量的值作为变量名

实现过程就是在变量的前面加一个$符号

自动类型转换：变量的类型由PHP自动转换，无需做任何操作

1、转换成布尔型

一些值会被转为false，除此之外，其他值会被转为true。具体如下（false）：

整型值0（零）

浮点型值 0.0（零）

空字符串，以及字符串 "0"

不包括任何元素的数组

不包括任何成员变量的对象

2、转换成整型

布尔型转换成整型：布尔值true，转换成整数1；布尔值false，转换成整数0。

强制类型转换： 在变量前加一个小括号，并把目标类型填写在括号中实现

（boolean）	强转为布尔型
（string）	强转为字符串型

(integer)　　强转为整型
(float)　　强转为浮点型
(array)　　强转为数
(object)　　强转为对象

赋值运算符

/=　除等于　　$a=3;$b=2;$a/=$b; $a=1.5;$b=2;
%=　模等于　　$a=3;$b=2;$a%=$b;　　$a=1;$b=2;
.=　连接等于　$a='abc';$a .= 'def';　　$a='abcdef'
在PHP语言中可以通过一条赋值语句对多个变量进行赋值
$a = $b = $c = 5;　　//为三个变量同时赋值

比较运算符

==　等于
!=　不等于
<>　不等于
===　　恒等
!==　不恒等

逻辑运算符

&&	与	$a && $b	$a和$b都为true，结果为true，否则为false
||	或	$a || $b	$a和$b中至少有一个为true，则结果为true，否则为false
!	非	!$a	若$a为false，结果为true，否则相反
xor	异或	$a xor $b	$a和$b其一为true，但不同时是，结果为true，否则为false
and	与	$a and $b	与&&相同，但优先级较低
or	或	$a or $b	与||相同，但优先级较低

错误控制运算符　　：使用@符号来表示，把它放在一个PHP表达式之前，将忽略该表达式可能产生的任何错误信息。

使用示例：
$a = @4/0;
注意：
@运算符只对表达式有效，例如可以把它放在变量、函数和include()调用、常量之前，但不能把它放在函数或类的定义之前。

流程控制语句

选择结构语句

if…elseif…else语句：执行语句用{}包含

switch语句：

```
switch (表达式){
        case 目标值1:
                执行语句1
                break;
        case 目标值2:
                执行语句2
                break;
        . . . . . .
        case 目标值n:
                执行语句n
                break;
        default:
                执行语句n+1
                break;
}
```

循环结构语句

1、while循环语句

2、do…while循环语句

3、for循环语句

```
for(初始化表达式; 循环条件; 操作表达式){
        执行语句
        ……
}
```

跳转语句

跳转语句用于实现循环执行过程中程序流程的跳转，在PHP中的跳转语句有

break语句、continue语句和goto语句

1、break语句

在switch条件语句和循环语句中都可以使用break语句。当它出现在switch条件语句中时，作用是终止某个case并跳出switch结构。当它出现在循环语句中，作用是跳出循环语句，执行后面的代码。

注意：break 可以接受一个可选的数字参数来决定跳出几重循环

2、continue语句：终止本次循环，执行下一次循环。

3、goto语句

注意：

goto语句仅在PHP 5.3及以上版本有效。

PHP中的goto语句只能在同一个文件或作用域中跳转，也就是说无法跳出一个函数或类方法，也无法跳入另一个函数。

函数

==========

函数的定义

function 函数名 ([参数1，参数2，……])

{

函数体

}

函数名不区分大小写，如search()和SEARCH()指的是同一个函数，这点与变量的命名不同。

函数的返回值

```
<?php
    function sum($a,$b)  {
      return $a+$b;
    }
     echo "两个数的和等于：";
     echo sum(23,96);
?>
```

函数中变量的作用域

```
<?php
    $var = 100;                          //此处$var是全局变量
    function test(&$i) {
   global $var;
    $i = 100+ $i+$var;
        echo "在函数内部var的值为：".$i;  //在函数内部调用全局变量$var，结果都是300
    }
    test($var);
     echo"<br>";
    echo"$var";
?>
```

上例中显示了：形参和实参、函数作用域、

&引用不是获得变量原本的值，而是指向原值。任何对引用的修改都会影响原变量值

global 关键字用于函数内访问全局变量。

可变函数

PHP 支持可变函数的概念，这意味着如果一个变量名后有圆括号，PHP 将寻找与变量的值同名的函数，并且尝试执行它。

```
<?php
    function calculatePrice($price,$discount){   //定义函数calculatePrice()
        $discount_price = $price * $discount;
          echo "商品的原价为".$price."元";
          echo "<br>";
        echo "商品的折扣为".$discount;
    echo "<br>";
        echo "商品的折扣价为".$discount_price."元";
        echo "<br>";
    }
    $price = 100;
    $discount = 0.7;
    calculatePrice($price,$discount);  //直接调用函数calculatePrice()
    $calculateFunc = "calculatePrice"; //将函数名"calculatePrice"赋值给变量$calculateFunc
    $calculateFunc($price,$discount); //调用与变量值同名的函数
?>
```

结果：

商品的原价为100元
商品的折扣为0.7
商品的折扣价为70元
商品的原价为100元
商品的折扣为0.7
商品的折扣价为70元

函数的嵌套调用

```
<?php
    function sum($subject1,$subject2,$subject3){          //定义计算总分的函数
            return $subject1 + $subject2 + $subject3;       //返回总分
    }
    function avg($subject1,$subject2,$subject3,$number){       //定义计算平均分的函数
            return sum($subject1,$subject2,$subject3) / $number;  //返回平均分
    }
    $chinese = 90;
```

```
    $math = 85;
    $english = 79;
    $number = 3;
    echo "平均分为" . avg($chinese,$math,$english,$number);
?>
```

结果：

平均分为84.666666666667

函数的递归调用

```
<?php
    // 下面的函数使用递归实现 求1~n的和
    function getSum($n) {

        if ($n == 1) {          // 满足条件，递归结束
            return 1;
        }
        $temp = getSum($n - 1);
        echo "<br/>";
        echo "$n";
        return $temp + $n;
    }
    echo "<br/>sum = ".getSum(4); // 调用递归函数，打印出1~4的和
?>
```

结果：

2

3

4

sum = 10

字符串相关函数

explode函数

作用：

使用一个字符串分割另一个字符串。每个元素都是 string 的一个子串

它们被字符串 $delimiter 作为边界点分割出来

函数：

explode (string $delimiter , string $string [, int $limit]) : array

参数：

$delimiter　　边界上的分隔字符。 $string　　输入要进行分割的字符串。

$limit　　如果设置limit参数并且是正数，则返回的数组包含最多limit个元素，而且最后哪个元素将包含string的剩余部分。

如果limit参数是负数，则返回最后的-limit个元素外的所有元素。

如果limit是0，则会当作1

返回值：

此函数返回由字符串组成的 array，每个元素都是 string 的一个子串，它们被字符串 delimiter 作为边界点分割出来。

备注：

如果 $delimiter 为空字符串（""），explode() 将返回 FALSE。

如果 $delimiter 所包含的值在 string 中找不到，并且使用了负数的 limit。

那么会返回空的 array， 否则返回包含 string 单个元素的数组。

例子：

```
<?php
    $str = 'tacks1 tacks2 tacks3 tacks4 tacks5';
var_dump(explode('',$str));//boolean false //空的$delimiter 会返回False并且报错。
var_dump(explode(' ',$str));//按照空格分开每个子串成为数组。
var_dump(explode(' ',$str,3));//数组元素为3个，前连两个按照指定的字符分割，剩下的全部挡在组后一个数组元素
var_dump(explode(' ',$str,-1));//在分割后的数组，删除最后一个元素tacks5
var_dump(explode('AAA',$str,0));//如果字符串中没有$delimiter，那么会全部当成数组的一个单元
?>
```

这里特别注意：**var_dump()** 函数用于输出变量的相关信息。

echo()函数是在屏幕上输出字符串，要输出列表数据需使用print_r() 函数

implode()函数

作用：

将一个一维数组的值按照特定的字符串$glue转化为字符串。

函数：

implode (string $glue , array $pieces) : string

join (string $glue , array $pieces) : string（join是implode的别名）

implode (array $pieces) : string (最好不用)

参数：

$glue 默认为空的字符串作为粘合数组每个元素 $pieces你想要转化的数组

返回值：

返回一个字符串，其内容为由 glue 分割开的数组的值。

备注：

因为历史原因，implode() 可以接收两种参数顺序，也就是第一个参数$glue也可以不写。

但是最好还是些两个参数向后兼容。而且第二个参数必须是一维数组。

例子：

```
$str = 'tacks1 tacks2 tacks3 tacks4 tacks5';
$arr = ['tacks1','tacks2','tacks3','tacks4','tacks5'];
echo implode(',',$arr),'<br/>';//tacks1,tacks2,tacks3,tacks4,tacks5
echo implode($arr),'<br/>';//tacks1tacks2tacks3tacks4tacks5
echo join('-',$arr),'<br/>';//tacks1-tacks2-tacks3-tacks4-tacks5
```

strcmp()函数

作用：判断两个字符串大小

函数：strcmp(string $str1,string $str2)

效果：返回一个整数

如果字符串$str1和$str2相等，则函数返回0；如果字符串$str1小于$str2，则函数返回值小于0（两个字符串的差值）；如果字符串$str1大于$str2，则函数返回值大于0。

例子：

```
echo strcmp("123","1234");  //-1
```

str_replace()函数

作用：字符串替换

函数：str_replace(string $search,string $replace,string $subject[,int &$count]) ： string

参数:

$search参数表示被替换掉的字符串，$replace参数表示替换后的字符串，$subject参数表示需要被操作的字符串，count()函数是用来统计$search参数被替换的次数，它是一个可选参数，与其他函数参数不同的是，当完成str_replace()函数的调用后，该参数还可以在函数外部直接被调用。

例子：

```
$sss = 'This is a book,That is an apple';
$str = 'apple';
echo str_replace('is',$str,$sss,$count),"<br/>",$count;
```

结果：

Thapple apple a book,That apple an apple

3

substr()函数

作用：截取一个字符串中的某一部分，也就是获取字符串中的某个子串

函数：string substr(string $str,int $start[,int $length])

参数: 函数名前的string表示函数的返回值类型是字符串类型，参数$str用于表示待处理的字符串，参数$start表示，从位置为start的字符处开始进行截取，参数$length表示截取的子串长度为

$length，该参数是可选的，如果$length为空，则默认截取到字符串的末尾。

例子：

```
$sss = 'This is a book,That is an apple';
echo substr($sss,1),"<br/>",substr($sss,0,1),"<br/>",substr($sss,-1);
```

结果：

his is a book,That is an apple

T

e

strlen()函数

作用：统计字符串的长度

函數：int strlen(string $str)

参数：int表示strlen()函数的返回值类型是整数类型，参数$str用于表示待获取长度的字符串。

例子：

```
$sss = 'This is a book,That is an apple';
echo strlen($sss); //31
```

trim()函数

作用：过滤字符串

函數：string trim (string $str [, string $charlist])

参数：函数名前的string表示函数的返回值类型是字符串类型，参数$str 用于表示待处理的字符串，参数$charlist是可选的，在调用函数时，若指定了$charlist，则函数会从字符串末端开始删除$charlist指定的字符，若没有指定$charlist，则函数会从字符串末端开始删除空白字符。

例子：

```
$sss = 'This     is     a       appleapple,That is an appleapple';
echo trim($sss,"apple"),"<br/>",trim($sss);
```

结果：

This is a appleapple,That is an //末尾的apple全部去除了

This is a appleapple,That is an appleapple//去除多余空格

日期和时间管理

获取时间

获取系统当前时间

time函数：

time(void)：int

time()函数没有参数，返回值为int类型。

date()函数

作用：格式化日期时间

参数：string date(string $format [, int $timestamp])

返回值是string类型，$timestamp为可选参数，如果没有指定则使用本地当前时间。format为固定参数，表示给定的格式

例子：

```
<?php
// Prints: July 1, 2000 is on a Saturday
echo "July 1, 2000 is on a " . date("l", mktime(0, 0, 0, 7, 1, 2000)),"<br>";
echo date("Y-m-d H:i:s",time()) ,"<br>";//当前时间
?>
```

关于date（）相关参数请参考：

https://www.runoob.com/php/php-date.html

PHP默认的时区设置是UTC，修改默认的时区设置，通常情况下有两种修改方式：

1.有权限情况下，直接修改php.ini中的date.timezone配置，例如将默认时区设置为PRC（中华人民共和国），具体如下：

date.timezone = PRC

修改完date.timezone配置后，需要重启服务器。

2.在程序中使用**date_default_timezone_set()**函数来设置时区，该函数的声明方式如下所示：

bool date_default_timezone_set(string $timezone_identifier)

例：date_default_timezone_set("PRC");

在上述声明中，返回类型是bool型，timezone_identifier用于指定时区标识符，可以是“PRC”、“Asia/Shanghai”或者“Asia/Chongqing”等

UNIX时间戳

Unix时间戳(Unix timestamp)是一种时间表示方式，定义为从格林威治时间**1970年01月01日00时00分00秒**起至现在的**总秒数。**以32位二进制数来表示，其中1970年1月1日零点也叫Unix纪元。时间戳不能为负数，因此1970年以前的时间戳无法使用

mktime()函数

作用：取得一个日期的 Unix 时间戳

参数：

mktime ([int $hour = date("H") [, int $minute = date("i") [, int $second = date("s") [, int $month = date("n") [, int $day = date("j") [, int $year = date("Y") [, int $is_dst = -1]]]]]]) : int

例子：

1：基本例子

```
<?php
// Set the default timezone to use. Available as of PHP 5.1
date_default_timezone_set('UTC');
// Prints: July 1, 2000 is on a Saturday
echo "July 1, 2000 is on a " . date("l", mktime(0, 0, 0, 7, 1, 2000));
// Prints something like: 2006-04-05T01:02:03+00:00
echo date('c', mktime(1, 2, 3, 4, 5, 2006));
?>
```

2：mktime() 在做日期计算和验证方面很有用，它会自动计算超出范围的输入的正确值

```
<?php
echo date("M-d-Y", mktime(0, 0, 0, 12, 32, 1997));
echo date("M-d-Y", mktime(0, 0, 0, 13, 1, 1997));
echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 1998));
echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 98));
?>
```

结果都为：Jan-01-1998

strtotime函数

作用：将时间字符串转换为时间或时间戳，经常用于获取用户提交的时间

参数：int strtotime(string $time [, int $now])

$time用于指定日期时间字符串，$now用于计算相对时间的参考点，如果省略则使用系统当前时间。

例子及用法：

```
<?php
echo strtotime("2016-1-22"),"<br>";//获取指定日期的unix时间戳
 /*获取英文文本日期时间,便于比较，使用date将当时间戳与指定时间戳转换成系统时间*/
echo date("Y-m-d H:i:s",time()) ,"<br>";//当前时间
echo date("Y-m-d H:i:s",strtotime("+1 day")) ,"<br>";/*明天此时的时间,打印明天此时的时间戳strtotime("+1 day")*/
echo date("Y-m-d H:i:s",strtotime("-1 day")),"<br>";/*昨天此时的时间，打印昨天此时的时间戳strtotime("-1 day") */
echo date("Y-m-d H:i:s",strtotime("+1 week")) ,"<br>";/*下个星期此时的时间,打印下个星期此时的时间戳strtotime("+1 week")*/
echo date("Y-m-d H:i:s",strtotime("-1 week")) ,"<br>";/*上个星期此时的时间，打印上个星期此时的时间戳strtotime("-1 week")*/
```

```
echo date("Y-m-d H:i:s",strtotime("next Thursday")) ,"<br>";/*指定下星期几的时间，打印指定下星期几的时间戳strtotime("next Thursday")*/
echo date("Y-m-d H:i:s",strtotime("last Thursday")) ,"<br>";/*指定上星期几的时间，打印指定上星期几的时间戳strtotime("last Thursday") */
?>
```

microtime()函数

作用：获取精确时间，如计算脚本的执行时间

参数：mixed microtime([bool $get_as_float])

该函数当前返回Unix时间戳以及微秒数，参数$get_as_float是可选参数，如果未设置为true将返回一个浮点数，如果省略，则以msec sec格式返回一个字符串，其中msec是微秒部分，sec是秒数，但都是以秒为单位返回的。

数组

数组类型：

索引数组：

索引数组的下标是从0开始，并依次递增

关联数组

关联数组是指下标为字符串的数组。它的键和值之间有一定的业务逻辑关系，因此，通常使用关联数组存储一系列具有逻辑关系的变量。

数组的定义：

1、使用赋值方式定义数组：$arrayName[key] = value

“$”是定义变量开始的标识符，“key”是数组的下标，其类型可以是整型或字符串，“value”可以是任意类型的数据。

2、使用array()函数定义数组：$arrayName = array(key1 => value1, key2 => value2, ...)

如果省略了key部分，则定义的数组默认为索引数组。

注意:

如果在定义数组时没有给某个元素指定下标，PHP就会自动将目前最大的那个整数下标值加1，作为该元素的下标，并依次递增后面元素的下标值。

数组元素的下标只有整型和字符串两种类型，如果是其他类型，则会进行类型转换。

由于合法的整型值的字符串下标会被类型转换为整型下标，所以在创建数组的时候，如果转换后数组存在相同的下标时，后面出现的元素值会覆盖前面的元素值。

数组的使用

$数组名[键名]

访问数组：可以使用方括号（[]）访问数组元素，还可以使用花括号（{}）。例如，$arr[0]和$arr{0}的效果是一样的。

输出数组：

PHP提供了print_r()和var_dump()函数，用于输出数组中的所有元素。

其中，print_r()函数可以按照一定格式显示数组中所有元素的键和值。

删除数组：

unset()函数

作用：删除数组中的元素。

数组指针

函数名	作用
mixed current (array &$array)	获取数组中当前元素的值，如果内部指针超出数组的末端，则返回false
mixed key (array &$array)	获取当前元素的下标，即键名
mixed next (array &$array)	将数组的内部指针向前移动一位
mixed prev (array &$array)	将数组的内部指针倒回一位
mixed end (array &$array)	将数组的内部指针指向最后一个元素
mixed reset (array &$array)	重置指针，即将数组的指针指向第一个元素

数组遍历

格式一：无键名遍历

```
foreach ($arr as $value) {
    循环体
}
```

格式二：键值对遍历

```
foreach ($arr as $key => $value) {
    循环体
}
```

两种语法格式中都是通过foreach语句来实现对数组的遍历，在格式一中，只是将当前元素的值赋给$value。而在格式二中，将当前元素的键名赋值给$key，值赋值给$value，这样可以同时获取当前元素的键名和值。

注意:

使用foreach遍历数组时，$key 和 $value 只不过是一个变量名而已，任何符合语法的变量名均可，如$k和$v。

$key和$value保存的数据是通过值传递的方式赋值的，这意味着对$key 和 $value的修改不影响数组本身。可以使用引用传递，在变量前加上&即可，但要注意这种方式只对 $value有效，$key不会改变。

例子：

```php
<?php
//定义数组
$numLists = array(1,2,3);
$numLists2 = array(1=>1,0=>2);
$numLists3[0] = '0';
$numLists4 = [5,2,4,7,9,4,2,6,8,3];
$getType = gettype($numLists3[0]);//查看数组元素类型
print_r($numLists3);
echo "<br/>",$getType,"<br/>";
unset($numLists[0]);//删除数组
print_r($numLists);
echo "<br/>==========================================<br/>";
//数组遍历
foreach ($numLists as $value) {
    echo $value," ";
}
echo "<br/>==========================================<br/>";
foreach ($numLists as $key => $value) {
    echo $key,"对应的值为：",$value,"<br/>";
}
?>
```

数组排序

冒泡排序：在冒泡排序的过程中，不断地比较数组中相邻的两个元素，较小者向上浮，较大者往下沉，整个过程和水中气泡上升的原理相似。

例子：

```php
<?php
//标准的冒泡排序
function paixu($arr)
 {
     $len = count($arr);
     for ($i = 0; $i < $len - 1; $i++) {//循环比对的轮数
         for ($j = $i + 1; $j < $len; $j++) {//从第二个开始循环，循环到最后一个，逐一和第一个比较
             if ($arr[$i] > $arr[$j]) {//前边大于后边的则交换
                 $tmp = $arr[$i];
                 $arr[$i] = $arr[$j];
                 $arr[$j] = $tmp;
```

```
            }
        }
    }
    return $arr;
}
$arr = [5,2,4,7,9,4,2,6,8,3];
print_r(paixu($arr));
//另外一种冒泡排序实现
echo "<br/>";
 function bubble_sort($arr)
{
   $len = count($arr);
    for ($i = 0; $i < $len -1; $i++) {//循环对比的轮数
        for ($j = 0; $j < $len - $i - 1; $j++) {//当前轮相邻元素循环对比
            if ($arr[$j] > $arr[$j + 1]) {//如果前边的大于后边的
                $tmp = $arr[$j];//交换数据
                $arr[$j] = $arr[$j + 1];
                 $arr[$j + 1] = $tmp;
            }
        }
    }
    return $arr;
}
$arr = [5,2,4,7,9,4,2,6,8,3];
print_r(bubble_sort($arr));
?>
```

数组元素查找

在数组中常用的查找元素的方法有顺序查找和二分法查找。

1、顺序查找法

顺序查找就是按照数组中的元素排列序号，从前往后一个一个查，如果找到则返回当前元素所在的下标。

2、二分查找法

二分法查找就是每次将指定元素和数组中间位置的元素进行比较，从而排除掉其中的一半元素，以此类推，继续进行查找。需要注意的是二分查找法只用于排序后的数组。

数组基本函数：

1、is_array()函数：作用是判断一个变量是否是数组，如果是数组，则返回true，否则返回false。

2、count()函数：作用是计算数组中元素的个数

声明方式：int count(mixed $var [, int $mode]);
在声明中，count()函数接收两个参数，其中$var参数是必需的，它表示传入的数组对象。$mode参数是可选参数，其值为0或1（COUNT_RECURSIVE）。该参数默认值为0，如果将该参数设置为1，则count()函数会递归计算多维数组中每个元素的个数。

3、array_unique()函数：作用是移除数组中的重复元素

array_unique()函数接收一个数组对象，去除重复元素后返回一个新的数组对象。

作用原理：在使用该函数时，首先将数组元素的值作为字符串排序，然后对每个值只保留第一个键名，忽略后面所有的键名。

数组键值对的相关函数：方便操作PHP数组键与值

1、array_search()函数：用于获取数组中元素对应的键名

声明方式：mixed array_search(mixed $needle , array $haystack [, bool $strict]);

$needle参数表示在数组中要查找的值，$haystack参数表示被查询的数组。$strict是可选参数，当值为true时，就会在$haystack数组中检查$needle的类型。

2、array_keys()函数：也是用于获取数组中元素对应的键名。不同的是，array_keys()函数可以返回所有匹配的键名

声明方式如下：

array array_keys(array $input[,mixed $search_value[,bool $strict]]);

$input参数表示被查询的数组。$search_value参数是可选参数，当给$search_value赋值时，该函数返回该值的键名，否则返回$input数组中的所有键名。自PHP 5起，可以用$strict参数来进行全等比较（===），需要传入一个布尔值，默认false，如果传入true值则根据类型返回带有指定值的键名。

数组排序函数：

sort()函数：对数组中的元素按照由小到大的顺序进行排序，

声明方式：bool sort (array &$array [, int $sort_flags = SORT_REGULAR]);

$array参数表示需要排序的数组，$sort_flags是可选参数，sort()函数会根据$sort_flag的值来改变数组的排序方式。

$sort_flag的取值范围以及对应的排序方式：

取值范围	**排序方式**
SORT_REGULAR	默认值，将自动识别数组元素的类型进行排序
SORT_NUMERIC	用于数字元素的排序
SORT_STRING	用于字符串元素的排序
SORT_LOCALE_STRING	根据当前的locale设置来把元素当做字符串比较

数组合并和拆分函数：

array_merge()函数：作用是合并一个或多个数组，

声明方式：array array_merge(array $array1 [, array $...]);

注意：

array_merge()将一个或多个数组的单元合并起来，一个数组中的值附加在另一个数组的后面，返回一个新的数组。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，而是附加到数组的后面。如果数组是数字索引的，则键名会以连续方式重新编排索引。

array_chunk()函数：作用是将一个数组分割成多个数组

声明方式：array array_chunk(array $input , int $size [, bool $preserve_keys]);

$input表示是要分割的数组，$size是分割后的每个数组中元素的个数。preserve_keys是一个可选参数，默认值为false，如果将该参数设置为true，则分割后的数组中元素保留原来的索引，如果将该参数设置为false，则分割后的数组中元素的索引将从零开始。

操作数组的其他函数：

array_rand()函数：作用是从数组中随机取出一个或多个元素

声明方式：mixed array_rand(array $input , int $num_req);

array_rand()函数接收一个input参数和一个参数num_req，其中input参数用于指定接收的数组，num_req参数用于指定取出元素的个数， 如果只取出一个元素，array_rand()会返回一个随机元素的键名，否则就返回一个包含随机键名的数组。

array_reverse()函数：作用是返回一个元素顺序相反的数组

声明方式：array array_reverse(array $array [, bool $preserve_keys]);

array_reverse()接收数组array作为输入并返回一个元素为相反顺序的新数组，如果preserve_keys为 true，则保留原来的键名。

例子：

```
<?php
//定义数组
$numLists = array(1,2,3,1,1,2,5);
$numLists2 = array_unique($numLists);
print_r($numLists2);
echo "<br/>==========================================<br/>";
echo array_search(1,$numLists),"<br/>";
print_r(array_keys($numLists,1)); //返回值为1的键名组成的数组
echo "<br/>==========================================<br/>";
//排序函数sort()
sort($numLists) ;
print_r($numLists);
echo "<br/>==========================================<br/>";
//数组合并与拆分
$numLists3 = array(6,7,8);
$numLists4 = array_merge($numLists,$numLists3);//数组合并
```

```
print_r($numLists4);
echo "<br/>==========<br/>";
print_r(array_chunk($numLists4,1)); //数组拆分
echo "<br/>==========<br/>";
print_r(array_rand($numLists,3)); //随机从数组中选择3个元素返回其键名组成的数组
echo " ",array_rand($numLists,1),"<br/> "; //随机从数组中选择1个元素返回其键名
print_r(array_reverse($numLists,true)); //加上第二个可选参数，true则保留元素之前的键名
?>
```

面向对象编程

面向对象的特点主要可以概括为封装性、继承性和多态性。

类的定义：

```
class 类名{
    成员属性;
    成员方法;
}
```

对象的创建：根据类创建实例对象，使用new关键字来创建对象

```
$对象名 = new 类名([参数1,参数2,...]);
```

类的封装：指在定义一个类时，将类中的属性私有化，即使用private关键字来修饰。私有化的属性只能在它所在类中被访问。

让外界访问私有属性，PHP提供了两种形式，具体如下：

1、通过getXxx()和setXxx()方法访问私有属性

为了可以访问私有属性，可以手动编写公有的getXxx()和setXxx()方法，其中，getXxx()方法用于获取属性值，setXxx()方法用于设置属性值。

2、通过__get()和__set ()方法访问私有属性

PHP5中预定义了__get()方法和__set()方法，__get()方法用于获取属性私有成员的属性值，__set()方法用于为私有成员属性赋值，两个方法获取私有属性和设置私有属性时都是自动调用的。

特殊的对象引用“$this”：它代表当前对象，用于完成对象内部成员之间的访问。

```
$this-> 属性名;
```

例子：

```
<?php
 class animal{
        private $name;
        private $color;
        private $age;
        public function __get($property_name) {
```

```
        echo "在直接获取私有属性值的时候，自动调用了这个__get()方法<br>";
        if(isset($this->$property_name)){
            return $this->$property_name;
        }
        else
        {
            return NULL;
        }
    }

    public function __set($propertyname, $value) {
        echo "在直接设置私有属性值的时候，自动调用了这个__set()方法为私有属性赋值<br>";
        $this->$propertyname = $value;
    }
}
$pig = new animal();
$pig->name = "猪";
$pig->color = "白色";
$pig->age = "1岁";
echo "称呼：".$pig->name."<br>";
echo "颜色：".$pig->color."<br>";
echo "年龄：".$pig->age."<br>";

?>
```

结果：

在直接设置私有属性值的时候，自动调用了这个__set()方法为私有属性赋值
在直接设置私有属性值的时候，自动调用了这个__set()方法为私有属性赋值
在直接设置私有属性值的时候，自动调用了这个__set()方法为私有属性赋值
在直接获取私有属性值的时候，自动调用了这个__get()方法
称呼：猪
在直接获取私有属性值的时候，自动调用了这个__get()方法
颜色：白色
在直接获取私有属性值的时候，自动调用了这个__get()方法
年龄：1岁

构造方法

每个类中，都有一个构造方法，在创建对象时会被自动调用。如果在类中没有显式的声明它，PHP会自动生成一个没有参数，且没有任何操作的默认构造方法。

修饰符 function __construct(参数列表){

  //初始化操作

}

需要注意的是构造方法的名称必须为__construct()，修饰符可以省略，默认为public

注意:

构造方法没有返回值。

构造方法的作用是完成对新对象的初始化，并不是创建对象本身。

在创建新对象后,系统会自动调用该类的构造方法，不需要手动调用。

一个类有且只有一个构造方法，在php5后虽然__construct()和类名()可以共存，但只能使用一个。

构造方法和普通方法一样，可以访问类属性和方法，也有访问控制修饰符，还可以被其它方法调用。

析构方法:PHP5中新添加的内容，它在对象销毁之前会被自动调用，用于释放内存

function __destruct(){

        //清理操作

}

需要注意的是，析构方法的名称必须为“__destruct()”，且析构方法不带任何参数。

例子:

```
class animal{
        //下面是动物的成员属性
        public $name = '';//成员属性name
        public $color = ''; //成员属性color
        public $age = ''; //成员属性size

        //定义一个构造方法， 参数为$name、$color和$age
        function __construct($name, $color, $age) {
            //通过构造方法传进来的$name 给成员属性$this->name赋初值
            $this->name = $name;
            //通过构造方法传进来的$color 给成员属性$this->color赋初值
            $this->color = $color;
            //通过构造方法传进来的$age 给成员属性$this->age赋初值
            $this->age = $age;
        }

        function getInfo(){
            echo '动物的名字叫做'.$this->name.'，动物的颜色是'.$this->color.'，动物的年龄是'.$this->age.'.';
```

```
        }
}
$pig = new animal("猪", "白色", "1岁");
//使用clone克隆新对象pig2,和$pig对象具有相同的属性和方法
$pig2 = clone $pig;
$pig2->getInfo();
```

类常量:属性的值不能改变，并且希望被所有对象所共享，例如圆周率，使用const关键字来申明.

const PI = 3.1415926; //定义一个常量属性PI

使用const关键字来声明常量，常量名前不需要添加$符号，并且在声明的同时必须对其进行初始化工作。

静态成员:可以实现类的所有对象共享一份数据

静态属性语法格式：访问修饰符 static 变量名

在语法格式中，static关键字写在访问修饰符的后面，访问修饰符可以省略，默认为public。

注意:

需要注意的是，静态属性是属于类而非对象，所以不能使用“对□属性”的方式来访问，而应该通过“类名::属性”的方式来访问，如果是在类的内部，还可以使用self关键字代替类名。

静态方法：在不创建对象的情况下就可以调用某个方法，也就是使该方法不必和对象绑在一起，可以使用静态方法

语法格式：访问修饰符 static 方法名(){}

静态方法的使用规则和静态属性相同，即通过类名称和范围解析操作符（::）来访问静态方法。

注意：

在静态方法中，不要使用$this。因为静态方法是属于类的，而$this则是指对象上下文。在静态方法中，一般只对静态属性进行操作

例子：

```
class MyClass{
        const constant = 'constant value';
        function showConstant(){
            echo self::constant."\n";
        }
    }
echo MyClass::constant,"<br/>";
$class = new MyClass();
$class->showConstant();
```

继承：

class 子类名 extends 父类名{
//类体
}
在PHP中只能实现单继承，也就是说子类只能继承一个父类（是指直接继承）。

重写父类方法：使用parent关键字

子类方法重写父类方法时，访问权限不能小于父类方法的访问权限。例如父类的方法是public的，在子类中重写时只能是public的。

final关键字：被final修饰的类和成员方法不能被修改。

1、final关键字修饰类

PHP中的类被final关键字修饰后，该类将不可以被继承，也就是不能够派生子类。

2、final关键字修饰方法

当一个类的方法被final关键字修饰后，这个类的子类将不能重写该方法。

自动加载：

从外部引入一个class，通常会使用include和require方法把定义这个class的文件包含进来。但是，在大型的开发项目中，这么做会产生大量的require或者include方法的调用，这样不仅会降低效率，而且使代码难以维护。如果不小心忘记引入某个类的定义文件，PHP就会报告一个致命错误，导致整个应用程序崩溃。

为解决此问题，PHP提供了类的自动加载机制，定义一个__autoload()函数，它会在试图使用尚未被定义的类时自动调用。这样，PHP在报告错误之前会有最后一个机会加载所需的类。

注意:

需要注意的是，自动加载是指当需要类定义文件而没有找到时，会自动的调用__autoload()函数，它不只限于实例化对象，还包括继承，序列化等操作。而且，自动加载并不能自己完成加载类的功能，它只提供了一个时机，具体的加载代码还需要用户编写代码实现。

魔术方法：

PHP中有很多以两个下划线开头的方法，如前面介绍的__construct()、__autoload()、__get()和__set()，这些方法被称为魔术方法。魔术方法有一个特点就是不需要手动调用，在某一时刻会自动执行，为程序的开发带来了极大的便利

方法声明	**功能描述**
__sleep()	对象序列化之前被调用，使程序延缓一段时间执行
__wakeup()	对象反序列化时被调用，还原被序列化的对象
__toString()	输出一个对象时被调用，将对象转化为字符串
__call()	在对象中调用一个不可访问方法时会被调用
__callStatic()	用静态方式中调用一个不可访问方法时被调用
__clone()	克隆对象时被调用
__invoke()	当尝试以调用函数的方式调用一个对象时被调用

抽象类：在定义一个类的时候，其中所需的某些方法暂时并不能完全定义出来，而是让其继承的类来实现

```
abstract class 类名{
        //类的成员
}
```

子类继承时再来编写该方法的具体实现。其语法格式如下：

```
abstract function 方法名();
```

注意:

抽象类不能被实例化。

抽象类可以没有抽象方法，但有抽象方法的抽象类才有意义。一旦类包含了抽象方法,则这个类必须声明为abstract。

抽象类中可以有非抽象方法, 成员属性和常量。

抽象方法不能有函数体，它只能存在于抽象类中。

如果一个类继承了某个抽象类，则它必须实现该抽象类的所有抽象方法，除非它自己也声明为抽象类。

接口：如果一个抽象类中的所有方法都是抽象的，则可以将这个类用另外一种方式来定义，即接口。

```
interface  Animal{
    public function run();
    public function breathe();
}
```

定义接口与定义类类似，但其中定义所有的方法都是空的。需要注意的是接口中的所有方法都是公有的，也不能使用final关键字来修饰。

由于接口中定义的都是抽象方法，没有具体实现，需要通过类来实现接口。实现接口使用implements关键字。

在使用implements关键字实现接口的同时，还可以使用extends关键字继承一个类。即在继承一个类的同时实现接口，但一定要先使用extends继承一个类，再使用implements实现接口，具体示例如下所示：

```
class 类名 extends 父类名 implements 接口1,接口2,……,接口n{
//实现所有接口中的抽象方法
}
```

命名空间：用于解决同名问题。同时还可以为标识符很长的名称创建别名，提高程序的可读性。

关键字namespace来声明：

```
<?php
   namespace MyProject;
```

```
    const CONNECT_OK = 1;
    class Connection { /* ... */ }
    function connect() { /* ... */  }
?>
```

命名空间的使用

PHP命名空间中类名可以通过以下三种方式引用：

非限定名称：即直接使用类名称，如$a = new foo()。它表示使用的是当前命名空间的foo类。

限定名称：即在类名前面添加前缀，如$a = new subnamespace\foo()。它表示使用的当前命名空间下子命名空间subnamespace下的foo类。

完全限定名称：即在类名前面添加命名空间前缀，如果有子命名空间也要写上，如$a = new \currentnamespace\foo()。它以命名空间反斜线开头的标识符，表示根命名空间下的子命名空间currentnamespace下的foo类。

总结：

self是类名的引用 ，始终与类class的当前类名绑定

$this 是当前类的实例的引用，他始终与当前类的实例绑定，用了$this就不用再实例化$obj=new XXX了。

Trait:1、代码复用的方式，用来扩展当前类的功能

2、当成一个公共方法库，

3、使用了类的定义的语法，但不是类，所以不能实例化。

在父类中用use 导入父类，即可使用

优先级，当前类中的同名方法>trait类中的同名方法>父类中的同名方法

接口：对象的模板是类，类的模板就是接口

面向接口编程是最重要的思想之一，很多高级应用都严重依附于它。

接口是一种约定，定义了实现他们的类中必须实现的方法。

接口中没有方法的具体实现，所以不能实例化。

用interface创建类接口，

抽象类 给其他类当父类

接口：全部都是抽象方法。

抽象类：抽象类中有有抽象方法，也有已实现的方法。

共同之处：统统不能实例化，原因就是内部有抽象方法。

错误处理及调试

常见错误类型

1、语法错误

2、运行错误

3、逻辑错误

4、环境错误：所需库并未加载

处理错误的两种方法：

标准PHP错误报告

标准PHP错误报告能够处理所有类型的错误，但是通常情况下，它只适用于PHP5之前的版本。

异常处理

异常处理被用于表示发生了一个异常事件并中断正常执行的脚本。

手动触发错误：根据不同的需求自定义错误

使用PHP的**内置函数trigger_error()来触发错误**，该函数声明如下：

bool trigger_error(string $error_msg [, int $error_type = E_USER_NOTICE])

上述声明中，第一个参数是错误信息内容，第二个参数是错误类别，默认为E_UESR_NOTICE。

显示错误报告

在PHP中实现显示错误的机制有以下几种方式：

1、修改配置文件

通过直接配置php.ini文件来实现显示错误报告，代码如下所示：

```
error_reporting(E_ALL & ~E_NOTICE);
display_errors = on;
```

上述代码中，error_reporting用于设置错误级别，display_errors用于设置是否显示错误报告。第1行代码中E_ALL & ~E_NOTICE表示显示除E_NOTICE之外的所有级别错误，第2行表示显示错误报告。

2、error_reporting()和ini_set()函数

通过PHP语言提供的error_reporting()和ini_set()函数来实现显示错误报告，代码如下所示：

```
<?php
    error_reporting(E_ALL & ~E_NOTICE);
    ini_set('display_errors',1);
?>
```

上述代码中，ini_set()函数用来设置错误信息是否显示，error_reporting()函数用于设置错误级别。第2行表示显示除E_NOTICE之外的所有级别错误，第3行表示显示错误信息。

3、die()函数

die()函数可以用来自定义输出错误信息，常用于业务逻辑的错误显示。

注意:使用函数控制的方式只对当前脚本有效，而配置php.ini文件对所有脚本都有效。

记录错误日志

1、修改配置文件

通过修改php.ini配置文件，可以直接设置记录错误日志的相关信息，具体代码如下所示：

error_reporting = E_ALL
log_error = On
error_log = /tmp/php_errors.log
上述代码中，error_reporting用于设置显示错误级别，E_ALL表示显示所有错误，log_error用于设置是否记录日志，error_log用于指定日志写入的文件路径。

2、error_log()函数

error_log()函数用于将错误记录到指定的日志文件中或发送电子邮件到指定地址，其函数声明如下：

bool error_log (string $message [, int $message_type = 0 [, string $destination [, string $extra_headers]]])

上述声明中，$message表示要记录的错误信息。参数$message_type表示消息类型，该参数有两个值0或1，0表示发送到服务器地址，1表示使用mail()函数发送到指定邮件地址。$destination表示错误日志记录的位置，$extra_headers表示额外的头，当$message_type=1时才会使用。

自定义错误处理器

自定义错误处理器是通过set_error_handler()函数来实现的，其函数声明如下：

mixed set_error_handler(callable $error_handler [, int $error_types = E_ALL | E_STRICT])

上述声明中，callable表示该参数$error_handler为回调函数类型。$error_handler是必须定义的参数，表示发生错误时运行的函数。$error_types用于指定处理错误的级别类型。

error_handler参数

在上述声明中error_handler参数必须符合错误处理器函数的原型，原型如下所示：

function handler(int $errno , string $errstr [, string $errfile [, int $errline [, array $errcontext]]]);

上述代码中，参数$errno表示错误级别，$errstr表示错误说明，$errfile表示发生错误代码的文件名称， errline表示错误发生的代码行的行号，$errcontext表示在触发错误的范围内存在的所有变量的数组。其中前两个参数是必填参数。

异常处理

PHP中可以通过throw关键字来抛出一个异常，如果要捕获和处理异常需要try...catch代码块来完成。

例子：

```
<?php
   //创建可抛出一个异常的函数
   function checkNum ($number){
         if($number > 1){
      //抛出异常
               throw new Exception("Value must be 1 or below");
         }  elseif($number = 1){
```

```
                trigger_error("数字不能为1");//php内置函数来触发错误
                }
        return true;
    }
    //可能触发异常的代码
    try{
        checkNum(2);
    }
    //捕获异常
    catch(Exception $e){
        echo 'Message: ' .$e->getMessage();
    }
?>
```

自定义异常

虽然PHP5提供的异常处理类Exception具备常用的一些功能。但有时候我们希望使用不同的异常类，针对特定类型的异常进行处理，此时就需要创建一个自定义异常类。

自定义异常类非常简单，只需要继承自Exception 类，并添加自定义的成员属性和方法即可。

设置顶层处理器

在实际开发中，为了保证程序正常运行，需要在所有可能出现异常的地方进行异常监视，但是程序出现异常的地方是无法预料的，为了保证程序的正常运行，PHP提供了set_exception_handler()函数用于对这些没有进行异常监视的异常进行处理，该函数也称为顶层异常处理器。

顶层异常处理器用于没有用try/catch块来捕获的异常，该函数声明如下所示：

callable set_exception_handler(callable $exception_handler)

PHP调试技术

使用输出函数进行调试

1、print() 和echo()

3、print_r()

4、var_dump():用于打印变量的相关信息

使用Debug工具，例如Xdebug