# 基本语法

## PHP标记

起始标记：<?php

结束标记：?>

这样写可以将php代码嵌入到任何的文档中

PHP也有短标记<? ?>,不鼓励；只有在php.ini中打开short\_open\_tag或者在编译是使用配置选项



如果文件内容是纯PHP代码，最好在文件末尾删除结束标记

版本5.4以后，增加了段标记：<?=

每个语句后面要用分号结束，如果是最后一行可以省略分号。

//最后一行可以不用使用分号，但要有php的结束标记

<?php

phpinfo()

?>

//如果没有结束标记，需要分号

<?php

phpinfo();

为什么不需要文件末尾的PHP结束标记：

文件末尾的 PHP 代码段结束标记可以不要，有些情况下当使用 [include](mk:@MSITStore:F:\Html\mannual_files_all\enhancingProgram\PHP\php_manual_zh_review.chm::/res/function.include.html) 或者 [require](mk:@MSITStore:F:\Html\mannual_files_all\enhancingProgram\PHP\php_manual_zh_review.chm::/res/function.require.html) 时省略掉会更好些，这样不期望的空白符就不会出现在文件末尾，之后仍然可以输出响应标头。在使用输出缓冲时也很便利，就不会看到由包含文件生成的不期望的空白符。

单行注释：// 或者 #

多行注释:/\* \*/,不能嵌套

PHP支持8中原始数据类型：

Boolean integer float/double string

Array object

Resource Null

伪类型：Mixed number callback

PHP中变量的类型通常不是由程序员设定的，是由PHP根据该变量使用的上下文在运行时决定的。

查看类型：var\_dump 或者 gettype 或者判断类型is\_null is\_string等

<?php

$str = 'abc';

$num = 12;

$nil = null;

var\_dump($str);

var\_dump($num);

var\_dump($nil);

echo gettype($str) . "\n";

echo gettype($num) . "\n";

echo gettype($nil) . "\n";

var\_dump(is\_int($str)) . "\n";

类型强制转换：settype 和各种前缀（int bool float string array object unset-转换为null）

<?php

//$str='addaa'; #强制转换为0

$str='12';

if(settype($str,'integer')){

var\_dump($str);

}else{

echo '强制转换失败';

}

### Boolean类型：

True和false关键字不区分大小写

<?php

$bol = TRUE;

$bol = false;

$bol = truE;

var\_dump($bol);

当运算符、函数或者控制流结构需要一个boolean参数时，会自动将参数转换为bool类型（可能是强制转换），以下值转换为boolean时，会被认为是flase：

false 0 0.0 ‘0’ 空字符串 空数组 不包含任何成员变量的对象（仅对PHP4.0适用） null 尚未赋值的变量 从空标记生成的simpleXML对象。

其他值都被认为是Ture，任何资源都为true

<?php

var\_dump((bool)"");

var\_dump((bool)"0");

var\_dump((bool)array());

var\_dump((bool)0);

var\_dump((bool)0.0);

var\_dump((bool)null);

### integer

整型值可以使用十进制、十六进制、八进制、二进制。在前面可以加上正负号。二进制自PHP 5.4.0起可用。八进制：数字前加0，十六进制：数字前加0x，二进制：数字前加0b

<?php

$a = 12; #十进制

var\_dump($a);

$b = 012; #八进制

var\_dump($b); //输出10，即输出时都以十进制表示

$c = 0x12; #十六进制

var\_dump($c);

$d = 0b11; #二进制

var\_dump($d);

$e = +12; #正数

var\_dump($e);

$f = -12; #负数

var\_dump($f);

整型数的字长和平台有关（不同的位数计算机表示的范围可能不同），PHP不支持无符号整数，自PHP 4.4.0he PHP 5.0.5后，可以使用PHP\_INT\_MAX来访问最大值。

echo 'PHP\_INT\_MAX: ' . PHP\_INT\_MAX;

如果一个整型数的范围超过了integer的范围，就会被解释为float

PHP中除法可以强制转换为integer或者使用函数round四舍五入。true转为1，false转为0；float取整时向下取整。

var\_dump(12/3); //int(4)

var\_dump(12/5); //double(2.4)

var\_dump((int)12/7); //double() 强制转换后为double

var\_dump(round(12/7)); //double(2)

var\_dump((int)true); // 1

var\_dump((int)false); // 0

不要将未知的分数强制转换为integer,如echo (int)((0.1+0.7)\*10);输出7

### float浮点型

浮点型也叫做浮点数float,双精度double或实数real，可以使用以下的方法来定义：

<?php

$a = 1.234;

var\_dump($a); //double(1.234)

$b = 1.2e+3;

var\_dump($b); //double(1200)

$d=1.2e3;

var\_dump($d); //double(1200)

$c=7e-2;

var\_dump($c); //double(0.07)

比较浮点数的两个方法：

直接比较两个浮点数时可能会有问题；如果要比较两个浮点数，需要使用一个最小误差值，该值也被称为机器极小值或最小单元取整数，作为判断两个浮点数是否相等时的允许误差。

1、直接比较

<?php

$a=1.23456781230;

$b = 1.23456781230;

var\_dump($a==$b);

2、设置允许误差

$a=1.23456781230;

$b = 1.23456781231;

var\_dump(abs($a-$b)<0.00001); //认为差值小于0.00001的两个数是相等的

NaN：not a number，代表着一个在浮点数运算中未定义或不可表述的值，与任何值的比较都是false，应该用is\_nan来判断。

var\_dump(is\_nan($a));

### String 字符串

PHP中每个字符就等同于一个字节（无论是汉字还是字母等符号），这意味着PHP只能支持256中字符集，从而不支持unicode

字符串的四种表达方式：

1. 单引号 2、双引号 3、heredoc结构 4、nowdoc结构

单引号：

在单引号中的变量和特殊字符的序列的转义将不会被替换

<?php

$a='abc';

$b='def';

var\_dump($a);

var\_dump('\t'); #这里的\t被当做两个字符

var\_dump("\t"); #这里的\t被当做一个字符-制表符

var\_dump('$b'); #这里直接输出$b

var\_dump("$b"); #这里输出def

双引号：

在双引号中，php将对一些特殊的字符进行解析，如\t、\n 等和变量；而且用双引号中的变量会被解析。

<?php

$a='abc';

$b='def';

var\_dump($a);

var\_dump('\t'); #这里的\t被当做两个字符

var\_dump("\t"); #这里的\t被当做一个字符-制表符

var\_dump('$b'); #这里直接输出$b

var\_dump("$b"); #这里输出def

heredoc结构：<<<标识符，标识符的定义规则与变量的定义规则一样，只能包含字母数字和下划线，并且必须以字母和下划线开头。

heredoc结构就像是没有使用双引号的双引号字符串，在其中的单引号不用被转义，变量将被替换。

<?php

$a='abc';

$b='def';

$c=<<<eof

this is a heredoc\tstruct

the 'first' variable $a ,

the second variable $b .

eof;

echo $c; //其中的$a和$b都被解析了,\t被解析了，‘没有被转义

自PHP 5.3.0起，还可以在heredoc结构中使用双引号来声明标识符，如：

<?php

$a='abc';

$b='def';

$c=<<<"eof"

this "is" a heredoc\tstruct

the 'first' variable $a ,

the second variable $b .

eof;

echo $c; //其中的$a和$b都被解析了,\t被解析了，‘没有被转义

nowdoc结构

就像heredoc结构类似于双引号字符串，nowdoc结构类似于单引号字符串。nowdoc中不进行解析操作（以下代码中的变量不会被解析，原样显示），结构表示法与heredoc大致相同，但标识符使用单引号括起来。

<?php

class foo{

public $foo;

public $bar;

public function foo(){

$this->foo='foo';

$this->bar=['bar1', 'bar2', 'bar3'];

}

}

$foo=new foo();

$name = 'myname';

$str=<<<’eof’

this is a class and the variatbles are

the first variable:$foo->foo

the second variale:$foo->bar

the variables in foo class are above!.

eof;

echo $str;

变量解析：

两种语法规则 1、简单语法 2、复杂语法

简单语法：当PHP解析器遇到一个美元符号$时，会和其他很多解析器一样，去组合尽可能多的表示以形成一个合法的变量名，可以使用花括号来明确变量名边界。

<?php

$a='apple';

echo "i have an $ab"; //PHP解析器解析变量$ab为空

echo 'i hava an $a'; //单引号内不解析

echo "i have an ${a} b"; //花括号限制了变量名的边界

复杂语法即花括号语法，可以使用复杂的表达式：

<?php

$a='fantastic';

echo "this is $a bcd\n"; #有效

echo "this is ${a} bcd\n"; #有效

echo "this is {$a} bcd\n"; #有效

$fantastic='haha';

echo "this is ${$a}"; //可变变量

$arr=['cc' => [1,2,3]];

echo "there is an item of array in this string {$arr['cc'][1]}"; #有效，字符串key要用单引号括起来

在花括号内可以使用对象

note：

函数、方法、静态类变量和类常量只有在PHP5以后才可以再{$}中使用，而且该字符串被定义的命名空间中才可以将其值作为变量名来访问，只使用单一花括号{}无法处理从函数或方法的返回值或者类常量以及类静态变量。

<?php

class beers{

const softdrink='rootbeer';

public $ale='ipa';

}

$rootbeer='A & W';

$ipa='alexander keith\'s';

$cc=new beers();

echo "i'd like an {${beers::softdrink}} \n"; //可变变量

echo "i'd like an {$cc->ale}";