|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.instruction-separation.html) | | |  | | | |
|  | **章 5. 基本语法**  **目录**  [从 HTML 中分离](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.html#language.basic-syntax.phpmode)  [指令分隔符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.instruction-separation.html)  [注释](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.comments.html)  **从 HTML 中分离**  PHP 的脚本由文件中一对特殊的标记所引起的内容，当解释器分析一个文件时，所有特殊标记外的普通文件，解释器不对其做任何处理，而特殊标记内文件将作为 PHP 代码被解释器分析执行。该运行机制将允许您在 HTML 中嵌入 PHP 代码：PHP 标记外的内容将被完全独立开，不做任何处理，而内部的将作为 PHP 代码解析。  用来标记 PHP 代码块的标记共有四套，其中只有两套总是有效（<?php. . .?> 和 <script language="php">. . .</script>）；另外两套可以在 php.ini 配置文件中开启或者关闭。虽然短格式的标记和 ASP 风格的标记可能比较省力，但是某些情况下您可能必须要使用长格式的标记。比如，您打算在 XML 或者 XHTML 中嵌入 PHP 代码，您将需要使用 <?php. . .?> 形式的标记以适应 XML 的标准。  PHP 支持的标记为：   |  |  | | --- | --- | | **例子 5-1. 从 HTML 中分离的方式**   |  | | --- | | 1.  <?php echo("if you want to serve XHTML or XML documents, do like this\n"); ?>  2.  <? echo ("this is the simplest, an SGML processing instruction\n"); ?>     <?= expression ?> This is a shortcut for "<? echo expression ?>"  3.  <script language="php">         echo ("some editors (like FrontPage) don't               like processing instructions");     </script>  4.  <% echo ("You may optionally use ASP-style tags"); %>     <%= $variable; # This is a shortcut for "<% echo . . ." %> | |   优先选用的方式为第一种方式 <?php. . .?>，因为它允许您在 XML 结构，如 XHTML 的代码中使用 PHP。  第二种方式并非总是可用的。只有当您在 php.ini 配置文件中激活 [short\_open\_tag](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.short-open-tag) 选项，才是可用的。您也可以通过使用 **short\_tags()** 函数（仅用于 PHP 3），或者使用 --enable-short-tags 选项 **configure** PHP 来激活短格式标记。在 php.ini-dist 配置文件中，默认是开启短格式标记的，但是建议您不要使用该标记。  在 PHP 配置文件中开启选项 [asp\_tags](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.asp-tags) 将激活第四种方式。  **注:** ASP 风格的标记添加于 PHP 3.0.4  **注:** 当开发大型应用程序或者用于分发的函数库，或者在不受您控制的 PHP 服务器上面开发程序时，请不要使用短格式的标记，因为目标服务器可能并不支持短格式的标记。为了便于移植，请保证用于再分发的代码中不使用短格式的标记。  PHP 代码块结束标记奖自动包含最近的一个结尾的新行（如果存在的话）。而且，结束标记自动隐含一个分号; 您不需要为 PHP 代码块的最后行追加一个分号。  PHP 允许您使用如下的结构：   |  |  | | --- | --- | | **例子 5-2. 更高级的脱离**   |  | | --- | | <?php if ($expression) {     ?>     <strong>This is true.</strong>     <?php } else {     ?>     <strong>This is false.</strong>     <?php } ?> | |   PHP 将直接输出结束标记和下一个开始标记中的任何非 PHP 代码。当需要输出大量的文本时，退出 PHP 解析模式将比使用 [**echo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.echo.html) 或者 [**print()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print.html) 或者此类的函数打印所有文本要更加的有效。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.instruction-separation.html) | | 语言参考 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 指令分隔符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.html) | 章 5. 基本语法 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.comments.html) | | |  | | | |
|  | **指令分隔符**  指令分隔方式与 C 或者 Perl 类似 -- 每个语句由分号格开。  结束标记（?>）同样隐含语句的结束，因此下面的代码是等价的：   |  | | --- | | <?php     echo "This is a test"; ?>  <?php echo "This is a test" ?> | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.comments.html) | | 基本语法 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.html) | 注释 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.instruction-separation.html) | 章 5. 基本语法 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | | |  | | | |
|  | **注释**  PHP 支持 'C'，'C++' 和 Unix Shell 风格的注释。例如:   |  | | --- | | <?php     echo "This is a test"; // This is a one-line c++ style comment     /\* This is a multi line comment        yet another line of comment \*/     echo "This is yet another test";     echo "One Final Test"; # This is shell-style style comment ?> |   “单行”注释仅仅注释到行末或者当前的 PHP 代码块，视乎哪个首先出现。   |  | | --- | | <h1>This is an <?php # echo "simple";?> example.</h1> <p>The header above will say 'This is an example'. |   小心不要嵌套 'C' 风格的注释，当注释大量代码时，可能犯该错误。   |  | | --- | | <?php /\*     echo "This is a test"; /\* This comment will cause a problem \*/ \*/ ?> |   “单行”注释仅仅注释到行末或者当前的 PHP 代码块，视乎哪个首先出现。这意味着 // ?> 后面的 HTML 代码将被打印出来：?> 跳出了 PHP 模式并且返回 HTML 模式，而且 // 注释符并不会影响到模式的转换。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.instruction-separation.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | | 指令分隔符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.html) | 类型 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.comments.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) | | |  | | | |
|  | 章 6. 类型目录[介绍](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html#language.types.intro)[布尔型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html)[整型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html)[浮点型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)[字符串](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)[数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html)[对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html)[资源](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html)[NULL](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html)[本文档中使用的伪类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html)[类型戏法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html)介绍PHP 支持八种原始类型。四种标量类型：布尔型（boolean）整型（integer）浮点型（float）（浮点数，也作“[double](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)”）字符串（string）两种复合类型：数组（array）对象（object）最后是两种特殊类型：资源（resource）[NULL](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html)为了确保代码的易读性，本手册还介绍了一些[伪类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html)：[混和（mixed）](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html#language.types.mixed)[数字（number）](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html#language.types.number)[回馈（callback）](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html#language.types.mixed)您可能还会读到一些关于“双精度（double）”类型的参考。实际上 double 和 float 是相同的，由于一些历史的原因，这两个名称同时存在。变量的类型通常不是由程序员设定的，确切地说，是由 PHP 根据该变量使用的上下文在运行时决定的。注: 如果你想查看某个[表达式](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html)的值和类型，用 [var\_dump()](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.var-dump.html)。注: 如果你只是想得到一个易读懂的类型的表达方式用于调试，用 [gettype()](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.gettype.html)。要查看某个类型，*不要*用 [gettype()](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.gettype.html)，而用 is\_*type* 函数。以下是一些范例：  |  | | --- | | <?php $bool = TRUE;   // a boolean $str  = "foo";  // a string $int  = 12;     // an integer  echo gettype($bool); // prints out "boolean" echo gettype($str);  // prints out "string"  // If this is an integer, increment it by four if (is\_int($int)) {     $int += 4; }  // If $bool is a string, print it out // (does not print out anything) if (is\_string($bool)) {     echo "String: $bool"; } ?> |   如果你要将一个变量强制转换为某类型，可以对其使用[强制转换](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html#language.types.typecasting)或者 [**settype()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.settype.html) 函数。  注意变量根据其当时的类型在特定场合下会表现出不同的值。更多信息见[类型戏法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html)。此外，你还可以参考 [PHP 类型比较表](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/types.comparisons.html)看不同类型相互比较的例子。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.basic-syntax.comments.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) | | 注释 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 布尔型 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) | | |  | | | |
|  | 布尔型 这是最简单的类型。[**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) 表达了真值，可以为 **TRUE** 或 **FALSE**。  **注:** 布尔类型是 PHP 4 引进的。 语法 要指定一个布尔值，使用关键字 **TRUE** 或 **FALSE**。两个都是大小写不敏感的。   |  | | --- | | <?php $foo = True; // assign the value TRUE to $foo ?> |   通常你用某些[运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html)返回 [**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) 值，并将其传递给[流程控制](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html)。   |  | | --- | | // == is an operator which test // equality and returns a boolean if ($action == "show\_version") {     echo "The version is 1.23"; }  // this is not necessary... if ($show\_separators == TRUE) {     echo "<hr>\n"; }  // ...because you can simply type if ($show\_separators) {     echo "<hr>\n"; } |  转换为布尔值 要明示地将一个值转换成 [**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html)，用 (bool) 或者 (boolean) 来强制转换。但是很多情况下不需要用强制转换，因为当运算符，函数或者流程控制需要一个 [**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) 参数时，该值会被自动转换。  参见[类型戏法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html)。  当转换为 [**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) 时，以下值被认为是 **FALSE**：   * [布尔](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html)值 **FALSE** * [整型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html)值 0（零） * [浮点型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)值 0.0（零） * 空白[字符串](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)和[字符串](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) "0" * 没有成员变量的[数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html) * 没有单元的[对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html) * 特殊类型[NULL](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html)（包括尚未设定的变量）   所有其它值都被认为是 **TRUE**（包括任何[资源](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html)）。   |  |  | | --- | --- | | **警告** | | | -1 和其它非零值（不论正负）一样，被认为是 **TRUE**！ | | | <?php echo gettype((bool) "");        // bool(false) echo gettype((bool) 1);         // bool(true) echo gettype((bool) -2);        // bool(true) echo gettype((bool) "foo");     // bool(true) echo gettype((bool) 2.3e5);     // bool(true) echo gettype((bool) array(12)); // bool(true) echo gettype((bool) array());   // bool(false) ?> | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) | | 类型 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 整型 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) | | |  | | | |
|  | 整型 一个 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 是集合 Z = {..., -2, -1, 0, 1, 2, ...} 中的一个数。  参见[任意长度整数 / GMP](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.gmp.html)，[浮点数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) 和 [任意精度数学库 / BCMath](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.bc.html)。 语法 整型值可以用十进制，十六进制或八进制符号指定，前面可以加上可选的符号（- 或者 +）。  如果用八进制符号，数字前必须加上 0（零），用十六进制符号数字前必须加上 0x。   |  |  | | --- | --- | | **例子 6-1. 整数文字表达**   |  | | --- | | <?php $a = 1234; # 十进制数 $a = -123; # 一个负数 $a = 0123; # 八进制数（等于十进制的 83） $a = 0x1A; # 十六进制数（等于十进制的 26） ?> | |   在字面上，整型变量正式的结构可以为：   |  | | --- | | <?php  decimal : [1-9][0-9]\*  | 0  hexadecimal : 0[xX][0-9a-fA-F]+  octal : 0[0-7]+  integer : [+-]?decimal  | [+-]?hexadecimal  | [+-]?octal  ?> |   整型数的字长和平台有关，尽管通常最大值是大约二十亿（32 位有符号）。PHP 不支持无符号整数。 整数溢出 如果你指定一个数超出了 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 的范围，将会被解释为 [**float**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)。同样如果你执行的运算结果超出了 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 范围，也会返回 [**float**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)。   |  |  | | --- | --- | | <?php $large\_number =  2147483647; var\_dump($large\_number); // 输出为：int(2147483647)  $large\_number =  2147483648; var\_dump($large\_number); // 输出为：float(2147483648)  // 同样也适用于十六进制表示的整数： var\_dump( 0x80000000 ); // 输出为：float(2147483648)  $million = 1000000; $large\_number =  50000 \* $million; var\_dump($large\_number); // 输出为：float(50000000000) ?> | | | **警告** | | | 不幸的是 PHP 中有个 bug，因此当有负数参与时结果并不总是正确。例如当运算 -50000 \* $million 时结果是 -429496728。不过当两个运算数都是正数时就没问题。  这个问题已经在 PHP 4.1.0 中解决了。 | |   PHP 中没有整除的运算符。1/2 产生出浮点数 0.5。您可以总是舍弃小数部分，或者使用 [**round()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.round.html) 函数。   |  | | --- | | <?php var\_dump(25/7);         // float(3.5714285714286) var\_dump((int) (25/7)); // int(3) var\_dump(round(25/7));  // float(4) ?> |  转换为整形 要明示地将一个值转换为 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html)，用 (int) 或 (integer) 强制转换。不过大多数情况下都不需要强制转换，因为当运算符，函数或流程控制需要一个 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 参数时，值会自动转换。您还可以通过函数 [**intval()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.intval.html) 来将一个值转换成整型。  参见[类型戏法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html)。 从[布尔值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html)转换 **FALSE** 将产生出 0（零），**TRUE** 将产生出 1（壹）。 从[浮点数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)转换 当从浮点数转换成整数时，数字将被取整（*丢弃小数位*）。  如果浮点数超出了整数范围（通常为 +/- 2.15e+9 = 2^31），则结果不确定，因为没有足够的精度使浮点数给出一个确切的整数结果。在此情况下没有警告，甚至没有任何通知！  （译者注：）在 Linux 下返回结果是最小负数（-2147483648），而在 Windows 下返回结果是零（0）。   |  | | --- | | **警告** | | 决不要将未知的分数强制转换为 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html)，这样有时会导致意外的结果。   |  | | --- | | <?php echo (int) ( (0.1+0.7) \* 10 ); // 显示 7！ ?> |   更多信息见[浮点数精度](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html#warn.float-precision)。 |  从字符串转换 参见[字符串转换为数字](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.conversion)。 从其它类型转换  |  | | --- | | **注意** | | 没有定义从其它类型转换为整型的行为。目前的行为和值先[转换为布尔值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html#language.types.boolean.casting)一样。不过*不要*依靠此行为，因为它会未加通知地改变。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) | | 布尔型 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 浮点型 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) | | |  | | | |
|  | 浮点型 浮点数（也叫“floats”，“doubles”或“real numbers”）可以用以下任何语法定义：   |  | | --- | | <?php $a = 1.234; $a = 1.2e3; $a = 7E-10; ?> |   形式上：   |  | | --- | | LNUM          [0-9]+ DNUM          ([0-9]\*[\.]{LNUM}) | ({LNUM}[\.][0-9]\*) EXPONENT\_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM}) |   浮点数的字长和平台相关，尽管通常最大值是 1.8e308 并具有 14 位十进制数字的精度（64 位 IEEE 格式）。   |  | | --- | | **浮点数精度** | | 显然简单的十进制分数如同 0.1 或 0.7 不能在不丢失一点点精度的情况下转换为内部二进制的格式。这就会造成混乱的结果：例如，floor((0.1+0.7)\*10) 通常会返回 7 而不是预期中的 8，因为该结果内部的表示其实是类似 7.9999999999...。  这和一个事实有关，那就是不可能精确的用有限位数表达某些十进制分数。例如，十进制的 1/3 变成了 0.3333333. . .。  所以永远不要相信浮点数结果精确到了最后一位，也永远不要比较两个浮点数是否相等。如果确实需要更高的精度，应该使用[任意精度数学函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.bc.html)或者 [gmp 函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.gmp.html)。 |  转换为浮点数 如果您希望了解有关何时和如何将字符串转换成浮点数的信息，请查阅标题为“[将字符串转换为数字](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.conversion)”的有关章节。对于其它类型的值，其情况类似于先将值转换成整型，然后再转换成浮点。请参阅“[转换为整型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html#language.types.integer.casting)”有关章节以获取更多信息。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) | | 整型 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 字符串 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html) | | |  | | | |
|  | 字符串 [**string**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) 是一系列字符。在 PHP 中，字符和字节一样，也就是说，一共有 256 种不同字符的可能性。这也暗示 PHP 对 Unicode 没有本地支持。请参阅函数 [**utf8\_encode()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.utf8-encode.html) 和 [**utf8\_decode()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.utf8-decode.html) 以了解有关 Unicode 支持。  **注:** 一个字符串变得非常巨大也没有问题，PHP 没有给字符串的大小强加实现范围，所以完全没有理由担心长字符串。 语法 字符串可以用三种字面上的方法定义。   * [单引号](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.syntax.single) * [双引号](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.syntax.double) * [定界符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.syntax.heredoc)  单引号 指定一个简单字符串的最简单的方法是用单引号（字符 '）括起来。  要表示一个单引号，需要用反斜线（\）转义，和很多其它语言一样。如果在单引号之前或字符串结尾需要出现一个反斜线，需要用两个反斜线表示。注意如果你试图转义任何其它字符，反斜线本身也会被显示出来！所以通常不需要转义反斜线本身。  **注:** 在 PHP 3 中，此情况下将发出一个 E\_NOTICE 级的警告。  **注:** 和其他两种语法不同，单引号字符串中出现的[变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html)和转义序列*不会*被变量的值替代。   |  | | --- | | <?php echo 'this is a simple string';  echo 'You can also have embedded newlines in strings this way as it is okay to do';  // Outputs: Arnold once said: "I'll be back" echo 'Arnold once said: "I\'ll be back"';  // Outputs: You deleted C:\\*.\*? echo 'You deleted C:\\\*.\*?';  // Outputs: You deleted C:\\*.\*? echo 'You deleted C:\\*.\*?';  // Outputs: This will not expand: \n a newline echo 'This will not expand: \n a newline';  // Outputs: Variables do not $expand $either echo 'Variables do not $expand $either'; ?> |  双引号 如果用双引号（"）括起字符串，PHP 懂得更多特殊字符的转义序列：  **表格 6-1. 转义字符**   | **序列** | **含义** | | --- | --- | | \n | 换行（LF 或 ASCII 字符 0x0A（10）） | | \r | 回车（CR 或 ASCII 字符 0x0D（13）） | | \t | 水平制表符（HT 或 ASCII 字符 0x09（9）） | | \\ | 反斜线 | | \$ | 美元符号 | | \" | 双引号 | | \[0-7]{1,3} | 此正则表达式序列匹配一个用八进制符号表示的字符 | | \x[0-9A-Fa-f]{1,2} | 此正则表达式序列匹配一个用十六进制符号表示的字符 |   此外，如果试图转义任何其它字符，反斜线本身也会被显示出来！  双引号字符串最重要的一点是其中的变量名会被变量值替代。细节参见[字符串解析](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.parsing)。 定界符 另一种给字符串定界的方法使用定界符语法（“<<<”）。应该在 <<< 之后提供一个标识符，然后是字符串，然后是同样的标识符结束字符串。  结束标识符*必须*从行的第一列开始。同样，标识符也必须遵循 PHP 中其它任何标签的命名规则：只能包含字母数字下划线，而且必须以下划线或非数字字符开始。   |  | | --- | | **警告** | | 很重要的一点必须指出，结束标识符所在的行不能包含任何其它字符，*可能*除了一个分号（;）之外。这尤其意味着该标识符*不能被缩进*，而且在分号之前和之后都不能有任何空格或制表符。同样重要的是要意识到在结束标识符之前的第一个字符必须是你的操作系统中定义的换行符。例如在 Macintosh 系统中是 \r。  如果破坏了这条规则使得结束标识符不“干净”，则它不会被视为结束标识符，PHP 将继续寻找下去。如果在这种情况下找不到合适的结束标识符，将会导致一个在脚本最后一行出现的语法错误。 |   定界符文本表现的就和双引号字符串一样，只是没有双引号。这意味着在定界符文本中不需要转义引号，不过仍然可以用以上列出来的转义代码。变量会被展开，但当在定界符文本中表达复杂变量时和字符串一样同样也要注意。   |  |  | | --- | --- | | **例子 6-2. 定界符字符串例子**   |  | | --- | | <?php $str = <<<EOD Example of string spanning multiple lines using heredoc syntax. EOD; /\* More complex example, with variables. \*/ class foo {     var $foo;     var $bar;      function foo()     {         $this->foo = 'Foo';         $this->bar = array('Bar1', 'Bar2', 'Bar3');     } }  $foo = new foo(); $name = 'MyName';  echo <<<EOT My name is "$name". I am printing some $foo->foo. Now, I am printing some {$foo->bar[1]}. This should print a capital 'A': \x41 EOT;?> | |   **注:** 定界符支持是 PHP 4 中加入的。 变量解析 当用双引号或者定界符指定字符串时，其中的[变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html)会被解析。  有两种语法，一种[简单的](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.parsing.simple)和一种[复杂的](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.parsing.complex)。简单语法最通用和方便，它提供了解析变量，数组值，或者对象属性的方法。  复杂语法是 PHP 4 引进的，可以用花括号括起一个表达式。 简单语法 如果遇到美元符号（$），解析器会尽可能多地取得后面的字符以组成一个合法的变量名。如果你想明示指定名字的结束，用花括号把变量名括起来。   |  | | --- | | <?php $beer = 'Heineken'; echo "$beer's taste is great"; // works, "'" is an invalid character for varnames echo "He drank some $beers";   // won't work, 's' is a valid character for varnames echo "He drank some ${beer}s"; // works echo "He drank some {$beer}s"; // works ?> |   同样也可以解析数组索引或者对象属性。对于数组索引，右方括号（]）标志着索引的结束。对象属性则和简单变量适用同样的规则，尽管对于对象属性没有像变量那样的小技巧。   |  | | --- | | <?php // These examples are specific to using arrays inside of strings. // When outside of a string, always quote your array string keys // and do not use {braces} when outside of strings either.  // Let's show all errors error\_reporting(E\_ALL);  $fruits = array('strawberry' => 'red', 'banana' => 'yellow'); // Works but note that this works differently outside string-quotes echo "A banana is $fruits[banana]."; // Works echo "A banana is {$fruits['banana']}.";  // Works but PHP looks for a constant named banana first // as described below. echo "A banana is {$fruits[banana]}.";  // Won't work, use braces.  This results in a parse error. echo "A banana is $fruits['banana'].";  // Works echo "A banana is " . $fruits['banana'] . ".";  // Works  echo "This square is $square->width meters broad."; // Won't work. For a solution, see the complex syntax. echo "This square is $square->width00 centimeters broad."; ?> |   对于任何更复杂的情况，应该使用复杂语法。 复杂（花括号）语法 不是因为语法复杂而称其为复杂，而是因为用此方法可以包含复杂的表达式。  事实上，用此语法你可以在字符串中包含任何在名字空间的值。仅仅用和在字符串之外同样的方法写一个表达式，然后用 { 和 } 把它包含进来。因为不能转义“{”，此语法仅在 $ 紧跟在 { 后面时被识别（用“{\$”或者“\{$”来得到一个字面上的“{$”）。用一些例子可以更清晰：   |  | | --- | | <?php // Let's show all errors error\_reporting(E\_ALL);  $great = 'fantastic';  // 不行，输出为：This is { fantastic} echo "This is { $great}";  // 可以，输出为：This is fantastic echo "This is {$great}"; echo "This is ${great}";  // Works echo "This square is {$square->width}00 centimeters broad.";  // Works echo "This works: {$arr[4][3]}"; // This is wrong for the same reason as $foo[bar] is wrong // outside a string.  In otherwords, it will still work but // because PHP first looks for a constant named foo, it will // throw an error of level E\_NOTICE (undefined constant). echo "This is wrong: {$arr[foo][3]}"; // Works.  When using multi-dimensional arrays, always use // braces around arrays when inside of strings echo "This works: {$arr['foo'][3]}";  // Works. echo "This works: " . $arr['foo'][3];  echo "You can even write {$obj->values[3]->name}";  echo "This is the value of the var named $name: {${$name}}"; ?> |  访问字符串中的字符 字符串中的字符可以通过在字符串之后用花括号指定所要字符从零开始的偏移量来访问。  **注:** 为了向下兼容，仍然可以用方括号。不过此语法在 PHP 4 中不赞成使用。   |  |  | | --- | --- | | **例子 6-3. 一些字符串例子**   |  | | --- | | <?php // Get the first character of a string $str = 'This is a test.'; $first = $str{0}; // Get the third character of a string $third = $str{2};  // Get the last character of a string. $str = 'This is still a test.'; $last = $str{strlen($str)-1}; ?> | |  实用函数及操作符 字符串可以用“.”（点）运算符连接。注意这里不能用“+”（加）运算符。更多信息参见[字符串运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html)。  有很多实用函数来改变字符串。  普通函数见[字符串函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.strings.html)一节，高级搜索和替换见正则表达式函数（两种口味：[Perl](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.pcre.html) 和 [POSIX 扩展](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.regex.html)）。  还有 [URL 字符串函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.url.html)，以及加密／解密字符串的函数（[mcrypt](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.mcrypt.html) 和 [mhash](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.mhash.html)）。  最后，如果还是找不到你要的函数，参见[字符类型函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.ctype.html)。 字符串转换 您可以用 (string) 标记或者 [**strval()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.strval.html) 函数将一个值转换为字符串。当某表达式需要字符串时，字符串的转换会在表达式范围内自动完成。例如当你使用 [**echo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.echo.html) 或者 [**print()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print.html) 函数时，或者将一个变量值与一个字符串进行比较的时候。阅读手册中有关[类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html)和[类型戏法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html)中的部分有助于更清楚一些。参见 [**settype()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.settype.html)。  布尔值 **TRUE** 将被转换为字符串 "1"，而值 **FALSE** 将被表示为 ""（即空字符串）。这样您就可以随意地在布尔值和字符串之间进行比较。  整数或浮点数数值在转换成字符串时，字符串即为表示这些数值数字（浮点数还包含有指数部分）。  数组将被转换成字符串 "Array"，因此您无法通过 [**echo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.echo.html) 或者 [**print()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print.html) 函数来输出数组的内容。请参考下文以获取更多提示。  对象将被转换成字符串 "Object"。如果您因为调试需要，需要将对象的成员变量打印出来，请阅读下文。如果您希望得到该对象所依附的类的名称，请使用函数 [**get\_class()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.get-class.html)。  资源类型将会以 "Resource id #1" 的格式被转换成字符串，其中 1 是 PHP 在运行时给资源指定的唯一标识。如果您希望获取资源的类型，请使用函数 [**get\_resource\_type()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.get-resource-type.html).  **NULL** 将被转换成空字符串。  正如以上所示，将数组、对象或者资源打印出来，并不能给您提供任何关于这些值本身的有用的信息。请参阅函数 [**print\_r()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print-r.html) 和 [**var\_dump()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.var-dump.html)，对于调试来说，这些是更好的打印值的方法。  您可以将 PHP 的值转换为字符串以永久地储存它们。这种方法被称为串行化，您可以用函数 [**serialize()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.serialize.html) 来完成该操作。如果您在安装 PHP 时建立了 [WDDX](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.wddx.html) 支持，您还可以将 PHP 的值串行化为 XML 结构。 字符串转换为数值 当一个字符串被当作数字来求值时，根据以下规则来决定结果的类型和值。  如果包括“.”，“e”或“E”其中任何一个字符的话，字符串被当作 [**float**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) 来求值。否则就被当作整数。  该值由字符串最前面的部分决定。如果字符串以合法的数字数据开始，就用该数字作为其值，否则其值为 0（零）。合法数字数据由可选的正负号开始，后面跟着一个或多个数字（可选地包括十进制分数），后面跟着可选的指数。指数是一个“e”或者“E”后面跟着一个或多个数字。   |  | | --- | | <?php $foo = 1 + "10.5";                // $foo is float (11.5) $foo = 1 + "-1.3e3";              // $foo is float (-1299) $foo = 1 + "bob-1.3e3";           // $foo is integer (1) $foo = 1 + "bob3";                // $foo is integer (1) $foo = 1 + "10 Small Pigs";       // $foo is integer (11) $foo = 4 + "10.2 Little Piggies"; // $foo is float (14.2) $foo = "10.0 pigs " + 1;          // $foo is float (11) $foo = "10.0 pigs " + 1.0;        // $foo is float (11) ?> |   此转换的更多信息见 Unix 手册中关于 strtod(3) 的部分。  如果你想测试本节中的任何例子，可以拷贝和粘贴这些例子并且加上下面这一行自己看看会发生什么：   |  | | --- | | <?php echo "\$foo==$foo; type is " . gettype ($foo) . "<br />\n"; ?> |   不要指望在将一个字符转换成整型时能够得到该字符的编码（您可能也会在 C 中这么做）。如果您希望在字符编码和字符之间转换，请使用 [**ord()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.ord.html) 和 [**chr()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.chr.html) 函数。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html) | | 浮点型 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 数组 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html) | | |  | | | |
|  | 数组 PHP 中的数组实际上是一个有序图。图是一种把 *values* 映射到 *keys* 的类型。此类型在很多方面做了优化，因此你可以把它当成真正的数组来使用，或列表（矢量），散列表（是图的一种实现），字典，集合，栈，队列以及更多可能性。因为可以用另一个 PHP 数组作为值，也可以很容易地模拟树。  解释这些结构超出了本手册的范围，但对于每种结构你至少会发现一个例子。要得到这些结构的更多信息，我们建议你参考有关此广阔主题的外部著作。 语法定义 [array()](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.array.html) 可以用 [**array()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.array.html) 语言结构来新建一个 [**array**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html)。它接受一定数量用逗号分隔的 *key* => *value* 参数对。   |  | | --- | | array( [*key* =>]  *value*  , ...  )  // *key* 可以是 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 或者 [**string**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)  // *value* 可以是任何值 |      |  | | --- | | <?php $arr = array("foo" => "bar", 12 => true);  echo $arr["foo"]; // bar echo $arr[12];    // 1 ?> |   key 可以是 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 或者 [**string**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)。如果键名是一个 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 的标准表达方法，则被解释为整数（例如 "8" 将被解释为 8，而 "08" 将被解释为 "08"）。PHP 中数组下标的变量类型不会对数组造成影响，数组的类型只有一种，它可以同时包含整型和字符串型的下标。  值可以是任何值。   |  | | --- | | <?php $arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));  echo $arr["somearray"][6];    // 5 echo $arr["somearray"][13];   // 9 echo $arr["somearray"]["a"];  // 42 ?> |   如果对给出的值没有指定键名，则取当前最大的整数索引值，而新的键名将是该值加一。如果你指定的键名已经有了值，则该值会被覆盖。   |  |  | | --- | --- | | <?php // This array is the same as ... array(5 => 43, 32, 56, "b" => 12);  // ...this array array(5 => 43, 6 => 32, 7 => 56, "b" => 12); ?> | | | **警告** | | | 自 PHP 4.3.0 起，上述的索引生成方法改变了。如今如果你给一个当前最大键名是负值的数组添加一个新值，则新生成的的索引将为零（0）。以前新生成的索引为当前最大索引加一，和正值的索引相同。 | |   使用 **TRUE** 作为键名将使 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 1 成为键名。使用 **FALSE** 作为键名将使 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 0 成为键名。使用 NULL 作为键名将等同于使用空字符串。使用空字符串作为键名将新建（或覆盖）一个用空字符串作为键名的值，这和用空的方括号不一样。  不能用数组和对象作为键名。这样做会导致一个警告：Illegal offset type。 用方括号的语法新建／修改 可以通过明示地设定值来改变一个现有的数组。  这是通过在方括号内指定键名来给数组赋值实现的。也可以省略键名，在这种情况下给变量名加上一对空的方括号（“[]”）。   |  | | --- | | $arr[*key*] = *value*;  $arr[] = *value*;  // *key* 可以是 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 或者 [**string**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)  // *value* 可以为任何值。 |   如果 $arr 还不存在，将会新建一个。这也是一种定义数组的替换方法。要改变一个值，只要给它赋一个新值。如果要删除一个键名／值对，要对它用 [**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html)。   |  | | --- | | <?php $arr = array(5 => 1, 12 => 2); $arr[] = 56;    // This is the same as $arr[13] = 56;                 // at this point of the script $arr["x"] = 42; // This adds a new element to                 // the array with key "x" unset($arr[5]); // This removes the element from the array unset($arr);    // This deletes the whole array ?> |   **注:** 如上所述，如果你给出方括号但没有指定键名，则取当前最大整数索引值，新的键名将是该值 + 1。如果当前还没有整数索引，则键名将为 0。如果制定的键名已经有值了，该值将被覆盖。   |  | | --- | | **警告** | | 自 PHP 4.3.0 起，上述的索引生成方法改变了。如今如果你给一个当前最大键名是负值的数组添加一个新值，则新生成的的索引将为零（0）。以前新生成的索引为当前最大索引加一，和正值的索引相同。 |   注意这里所使用的最大整数键名*不一定当前就在数组中*。它只要在上次数组重新生成索引后曾经存在过就行了。以下例子说明了：   |  | | --- | | <?php // 创建一个简单的数组 $array = array(1, 2, 3, 4, 5); print\_r($array);  // 现在删除其中的所有单元，但保持数组本身的结构 foreach ($array as $i => $value) {     unset($array[$i]); } print\_r($array);  // 添加一个单元（注意新的键名是 5，而不是你可能以为的 0） $array[] = 6; print\_r($array);  // 重新索引： $array = array\_values($array); $array[] = 7; print\_r($array); ?> |   以上例子将产生如下输出：   |  | | --- | | Array  (  [0] => 1  [1] => 2  [2] => 3  [3] => 4  [4] => 5  )  Array  (  )  Array  (  [5] => 6  )  Array  (  [0] => 6  [1] => 7  ) |  实用函数 有相当多的实用函数作用于数组，参见[数组函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.array.html)一节。  **注:** [**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html) 函数允许取消一个数组中的键名。要注意数组将不会重建索引。   |  | | --- | | <?PHP $a = array( 1 => 'one', 2 => 'two', 3 => 'three' ); unset( $a[2] ); /\* 将产生一个数组，定义为    $a = array( 1=>'one', 3=>'three');    而不是    $a = array( 1 => 'one', 2 => 'three'); \*/ $b = array\_values($a); // Now b is array(1 => 'one', 2 =>'three') ?> |   [foreach](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.foreach.html) 控制结构是专门用于数组的。它提供了一个简单的方法来遍历数组。 数组做什么和不做什么为什么 $foo[bar] 错了？ 应该始终在用字符串表示的数组索引上加上引号。例如用 $foo['bar'] 而不是 $foo[bar]。但是为什么 $foo[bar] 错了呢？你可能在老的脚本中见过如下语法：   |  | | --- | | <?php $foo[bar] = 'enemy'; echo $foo[bar]; // etc ?> |   这样是错的，但可以正常运行。那么为什么错了呢？原因是此代码中有一个未定义的常量（bar）而不是字符串（'bar'－注意引号），而 PHP 可能会在以后定义此常量，不幸的是你的代码中有同样的名字。它能运行，是因为 PHP 自动将*裸字符串*（没有引号的字符串且不对应于任何已知符号）转换成一个其值为该裸字符串的正常字符串。例如，如果没有常量定义为 **bar**，PHP 将把它替代为 'bar' 并使用之。  **注:** 这并不意味着*总是*给键名加上引号。用不着给键名为[常量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) 或 [变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) 的加上引号，否则会使 PHP 不能解析它们。   |  | | --- | | <?php error\_reporting(E\_ALL); ini\_set('display\_errors', true); ini\_set('html\_errors', false); // Simple array: $array = array(1, 2); $count = count($array); for ($i = 0; $i < $count; $i++) {     echo "\nChecking $i: \n";     echo "Bad: " . $array['$i'] . "\n";     echo "Good: " . $array[$i] . "\n";     echo "Bad: {$array['$i']}\n";     echo "Good: {$array[$i]}\n"; } ?> |   **注:** 上面例子输出为：   |  | | --- | | Checking 0:  Notice: Undefined index: $i in /path/to/script.html on line 9  Bad:  Good: 1  Notice: Undefined index: $i in /path/to/script.html on line 11  Bad:  Good: 1  Checking 1:  Notice: Undefined index: $i in /path/to/script.html on line 9  Bad:  Good: 2  Notice: Undefined index: $i in /path/to/script.html on line 11  Bad:  Good: 2 |   演示此效应的更多例子：   |  | | --- | | <?php // 显示所有错误 error\_reporting(E\_ALL);  $arr = array('fruit' => 'apple', 'veggie' => 'carrot');  // 正确 print $arr['fruit'];  // apple print $arr['veggie']; // carrot  // 不正确。This works but also throws a PHP error of // level E\_NOTICE because of an undefined constant named fruit // // Notice: Use of undefined constant fruit - assumed 'fruit' in... print $arr[fruit];    // apple  // Let's define a constant to demonstrate what's going on.  We // will assign value 'veggie' to a constant named fruit. define('fruit','veggie');  // Notice the difference now print $arr['fruit'];  // apple print $arr[fruit];    // carrot  // The following is okay as it's inside a string.  Constants are not // looked for within strings so no E\_NOTICE error here print "Hello $arr[fruit]";      // Hello apple  // With one exception, braces surrounding arrays within strings // allows constants to be looked for print "Hello {$arr[fruit]}";    // Hello carrot print "Hello {$arr['fruit']}";  // Hello apple  // This will not work, results in a parse error such as: // Parse error: parse error, expecting T\_STRING' or T\_VARIABLE' or T\_NUM\_STRING' // This of course applies to using autoglobals in strings as well print "Hello $arr['fruit']"; print "Hello $\_GET['foo']";  // Concatenation is another option print "Hello " . $arr['fruit']; // Hello apple ?> |   当打开 [**error\_reporting()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.error-reporting.html) 来显示 **E\_NOTICE** 级别的错误（例如将其设为 **E\_ALL**）时将看到这些错误。默认情况下 [error\_reporting](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.errorfunc.html#ini.error-reporting) 被关闭不显示这些。  和在[语法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html#language.types.array.syntax)一节中规定的一样，在方括号（“[”和“]”）之间必须有一个表达式。这意味着你可以这样写：   |  | | --- | | <?php echo $arr[somefunc($bar)]; ?> |   这是一个用函数返回值作为数组索引的例子。PHP 也可以用已知常量，你可能之前已经见过 E\_\*。   |  | | --- | | <?php $error\_descriptions[E\_ERROR]   = "A fatal error has occured"; $error\_descriptions[E\_WARNING] = "PHP issued a warning"; $error\_descriptions[E\_NOTICE]  = "This is just an informal notice"; ?> |   注意 E\_ERROR 也是个合法的标识符，就和第一个例子中的 bar 一样。但是上一个例子实际上和如下写法是一样的：   |  | | --- | | <?php $error\_descriptions[1] = "A fatal error has occured"; $error\_descriptions[2] = "PHP issued a warning"; $error\_descriptions[8] = "This is just an informal notice"; ?> |   因为 E\_ERROR 等于 1，等等。  如同我们在以上例子中解释的那样，$foo[bar] 起作用但其实是错误的。它起作用是因为根据语法的预期，bar 被当成了一个常量表达式。然而，在这个例子中不存在名为 bar 的常量。PHP 就假定你指的是字面上的 bar，也就是字符串 "bar"，但你忘记写引号了。 那么为什么这样做不好？ 在未来的某一时刻，PHP 开发小组可能会想新增一个常量或者关键字，或者您可能希望在以后在您的程序中引入新的常量，那你就有麻烦了。例如你已经不能这样用 empty 和 default 这两个词了，因为他们是[保留字](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.html)。  **注:** 重申一次，在双引号字符串中，不给索引加上引号是合法的因此 "$foo[bar]"是合法的。至于为什么参见以上的例子和[字符串中的变量解析](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.parsing)中的解释。 转换为数组 对于任何的类型：整型、浮点、字符串、布尔和资源，如果您将一个值转换为数组，您将得到一个仅有一个元素的数组（其下标为 0），该元素即为此标量的值。  如果您将一个对象转换成一个数组，您所得到的数组的元素为该对象的属性（成员变量），其键名为成员变量名。  如果您将一个 **NULL** 值转换成数组，您将得到一个空数组。 例子 PHP 中的数组类型有非常多的用途，因此这里有一些例子展示数组的完整威力。   |  | | --- | | <?php // this $a = array( 'color' => 'red',             'taste' => 'sweet',             'shape' => 'round',             'name'  => 'apple',                        4        // key will be 0           );  // is completely equivalent with $a['color'] = 'red'; $a['taste'] = 'sweet'; $a['shape'] = 'round'; $a['name']  = 'apple'; $a[]        = 4;        // key will be 0  $b[] = 'a'; $b[] = 'b'; $b[] = 'c'; // will result in the array array(0 => 'a' , 1 => 'b' , 2 => 'c'), // or simply array('a', 'b', 'c') ?> | | **例子 6-4. 使用 array()**   |  | | --- | | <?php // Array as (property-)map $map = array( 'version'    => 4,               'OS'         => 'Linux',               'lang'       => 'english',               'short\_tags' => true             );  // strictly numerical keys $array = array( 7,                 8,                 0,                 156,                 -10               ); // this is the same as array(0 => 7, 1 => 8, ...)  $switching = array(         10, // key = 0                     5    =>  6,                     3    =>  7,                     'a'  =>  4,                             11, // key = 6 (maximum of integer-indices was 5)                     '8'  =>  2, // key = 8 (integer!)                     '02' => 77, // key = '02'                     0    => 12  // the value 10 will be overwritten by 12                   ); // empty array $empty = array(); ?> | |  |  |  | | --- | --- | | **例子 6-5. 集合**   |  | | --- | | <?php $colors = array('red', 'blue', 'green', 'yellow');  foreach ($colors as $color) {     echo "Do you like $color?\n"; }  /\* output: Do you like red? Do you like blue? Do you like green? Do you like yellow? \*/ ?> | |   注意目前不可能在这样一个循环中直接改变数组的值。可以改变的例子如下：   |  |  | | --- | --- | | **例子 6-6. 集合**   |  | | --- | | <?php foreach ($colors as $key => $color) {     // won't work:     //$color = strtoupper($color);      //works:     $colors[$key] = strtoupper($color); } print\_r($colors);  /\* output: Array (     [0] => RED     [1] => BLUE     [2] => GREEN     [3] => YELLOW ) \*/ ?> | |   本例产生一个基于一的数组。   |  |  | | --- | --- | | **例子 6-7. 基于一的数组**   |  | | --- | | <?php $firstquarter  = array(1 => 'January', 'February', 'March'); print\_r($firstquarter); /\* output: Array (     [1] => 'January'     [2] => 'February'     [3] => 'March' ) \*/ ?> | |  |  |  | | --- | --- | | **例子 6-8. 填充数组**   |  | | --- | | <?php // fill an array with all items from a directory $handle = opendir('.'); while (false !== ($file = readdir($handle))) {     $files[] = $file; } closedir($handle); ?> | |   数组是有序的。你也可以使用不同的排序函数来改变顺序。更多信息参见[数组函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.array.html)。您可以用 [**count()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.count.html) 函数来数出数组中元素的个数。   |  |  | | --- | --- | | **例子 6-9. 数组排序**   |  | | --- | | <?php sort($files); print\_r($files); ?> | |   因为数组中的值可以为任意值，也可是另一个数组。这样你可以产生递归或多维数组。   |  |  | | --- | --- | | **例子 6-10. 递归和多维数组**   |  | | --- | | <?php $fruits = array ( "fruits"  => array ( "a" => "orange",                                        "b" => "banana",                                        "c" => "apple"                                      ),                   "numbers" => array ( 1,                                        2,                                        3,                                        4,                                        5,                                        6                                      ),                   "holes"   => array (      "first",                                        5 => "second",                                             "third"                                      )                 );  // Some examples to address values in the array above echo $fruits["holes"][5];    // prints "second" echo $fruits["fruits"]["a"]; // prints "orange" unset($fruits["holes"][0]);  // remove "first"  // Create a new multi-dimensional array $juices["apple"]["green"] = "good"; ?> | |   您需要注意数组的赋值总是会涉及到值的拷贝。您需要在复制数组时用指向符号（&）。   |  | | --- | | <?php $arr1 = array(2, 3); $arr2 = $arr1; $arr2[] = 4; // $arr2 is changed,              // $arr1 is still array(2,3)  $arr3 = &$arr1; $arr3[] = 4; // now $arr1 and $arr3 are the same ?> | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html) | | 字符串 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 对象 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html) | | |  | | | |
|  | 对象对象初始化 要初始化一个对象，用 new 语句将对象实例到一个变量中。   |  | | --- | | <?php class foo {     function do\_foo()     {         echo "Doing foo.";     } }  $bar = new foo; $bar->do\_foo(); ?> |   完整的讨论见[类与对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html)一章。 转换为对象 如果将一个对象转换成对象，它将不会有任何变化。如果其它任何类型的值被转换成对象，内置类 stdClass 的一个实例将被建立。如果该值为 **NULL**，则新的实例为空。对于任何其它的值，名为 scalar 的成员变量将包含该值。   |  | | --- | | <?php $obj = (object) 'ciao'; echo $obj->scalar;  // outputs 'ciao' ?> | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html) | | 数组 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 资源 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) | | |  | | | |
|  | 资源 一个资源是一个特殊变量，保存了到外部资源的一个引用。资源是通过专门的函数来建立和使用的。所有这些函数及其相应资源类型见[附录](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/resource.html)。  **注:** 资源类型是 PHP 4 引进的。 转换为资源 由于资源类型变量保存有为打开文件、数据库连接、图形画布区域等的特殊句柄，您无法将其它类型的值转换为资源。 释放资源 由于 PHP4 Zend 引擎引进了资源计数系统，可以自动检测到一个资源不再被引用了（和 Java 一样）。这种情况下此资源使用的所有外部资源都会被垃圾回收系统释放。由此原因，很少需要用某些 free-result 函数来手工释放内存。  **注:** 持久数据库连接比较特殊，它们*不会*被垃圾回收系统破坏。参见[数据库永久连接](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.persistent-connections.html)一章。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) | | 对象 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | NULL | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html) | | |  | | | |
|  | NULL 特殊的 **NULL** 值表示一个变量没有值。[**NULL**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) 类型唯一可能的值就是 **NULL**。  **注:** [**NULL**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) 类型是 PHP 4 引进的。  在下列情况下一个变量被认为是 **NULL**：   * 被赋值为 **NULL**。 * 尚未被赋值。 * 被 [**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html)。  语法 **NULL** 类型只有一个值，就是大小写敏感的关键字 **NULL**。   |  | | --- | | <?php $var = NULL; ?> |   参见 [**is\_null()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-null.html) 和 [**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html)。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html) | | 资源 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 本文档中使用的伪类型 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html) | | |  | | | |
|  | 本文档中使用的伪类型mixed 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。  [**gettype()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.gettype.html) 表明可以接受所有的 PHP 类型，例如 [**str\_replace()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.str-replace.html) 将接受字符串和数组。 number number 说明一个参数可以是 [**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html) 或者 [**float**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html)。 callback 有些诸如 **call\_user\_function()** 或 [**usort()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.usort.html) 的函数接受用户自定义的函数作为一个参数。Callback 函数不仅可以是一个简单的函数，它还可以是一个对象的方法，包括静态类的方法。  一个 PHP 函数用函数名字符串来传递。您可以传递任何内建的或者用户自定义的函数，除了 [**array()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.array.html)， [**echo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.echo.html)， [**empty()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.empty.html)， [**eval()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.eval.html)， [**exit()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.exit.html)， [**isset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.isset.html)， [**list()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.list.html)， [**print()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print.html) 和 [**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html)。  一个对象的方法以数组的形式来传递，数组的 0 下标指明对象名，下标 1 指明方法名。  对于没有实例化为对象的静态类，要传递其方法，将数组 0 下标指明的对象名换成该类的名称即可。   |  |  | | --- | --- | | **例子 6-11. Callback 函数实例**   |  | | --- | | <?php  // simple callback example function my\_callback\_function() {     echo "hello world!"; } call\_user\_function("my\_callback\_function");  // method callback examples class MyClass {   function myCallbackMethod() {     echo "hello world!";   } } // static class method call without instantiating an object call\_user\_func(array('MyClass', 'myCallbackMethod')); // object method call $obj = new MyClass(); call\_user\_func(array(&$obj, 'myCallbackMethod')); ?> | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.null.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html) | | NULL | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 类型戏法 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html) | 章 6. 类型 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | | |  | | | |
|  | 类型戏法 PHP 在变量定义中不需要（或不支持）明示的类型定义；变量类型是根据使用该变量的上下文所决定的。也就是说，如果你把一个字符串值赋给变量 *var*，*var* 就成了一个字符串。如果你又把一个整型值赋给 *var*，那它就成了一个整数。  PHP 的自动类型转换的一个例子是加号“+”。如果任何一个运算数是浮点数，则所有的运算数都被当成浮点数，结果也是浮点数。否则运算数会被解释为整数，结果也是整数。注意这并没有改变这些运算数本身的类型；改变的仅是这些运算数如何被求值。   |  | | --- | | <?php $foo = "0";  // $foo is string (ASCII 48) $foo += 2;   // $foo is now an integer (2) $foo = $foo + 1.3;  // $foo is now a float (3.3) $foo = 5 + "10 Little Piggies"; // $foo is integer (15) $foo = 5 + "10 Small Pigs";     // $foo is integer (15) ?> |   如果上面两个例子看上去古怪的话，参见[字符串转换为数值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.conversion)。  如果你要强制将一个变量当作某种类型来求值，参见[类型强制转换](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html#language.types.typecasting)一节。如果你要改变一个变量的类型，参见 [**settype()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.settype.html)。  如果你想要测试本节中任何例子的话，可以用 [**var\_dump()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.var-dump.html) 函数。  **注:** 数组的自动转换行为目前没有定义。   |  | | --- | | <?php $a = "1";     // $a 是字符串 $a[0] = "f";  // 是字符串偏移量吗？结果会是什么？ ?> |   由于一些历史原因，PHP 支持通过偏移量进行的字符串索引，这和数组索引的语法一样。以上的例子就产生了一个问题：$a 应该变成一个第一个元素是“f”的数组呢，还是“f”成了字符串 $a 的第一个字符？  目前版本的 PHP 将以上第二个赋值理解成字符串的偏移量标识，即 $a 变成了 "f"，尽管如此，这种自动转换的地结果应该被认为未定义。PHP 4 引入了新的花括号语法来访问字符串的字符，请使用该语法来替代以上的操作：   |  | | --- | | <?php $a    = "abc"; // $a 为一个字符串 $a{1} = "f";   // $a 目前为 "afc" ?> |   请参阅题为“[用字符访问字符串](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.substr)”的章节以获取更多信息。 类型强制转换 PHP 中的类型强制转换和 C 中的非常像：在要转换的变量之前加上用括号括起来的目标类型。   |  | | --- | | <?php $foo = 10;   // $foo is an integer $bar = (boolean) $foo;   // $bar is a boolean ?> |   允许的强制转换有：   * (int), (integer) - 转换成整型 * (bool), (boolean) - 转换成布尔型 * (float), (double), (real) - 转换成浮点型 * (string) - 转换成字符串 * (array) - 转换成数组 * (object) - 转换成对象   注意在括号内允许有空格和制表符，所以下面两个例子功能相同：   |  | | --- | | <?php $foo = (int) $bar; $foo = ( int ) $bar; ?> |   **注:** 为了将一个变量还原为字符串，您还可以将变量放置在双引号种。   |  | | --- | | <?php $foo = 10;            // $foo is an integer $str = "$foo";        // $str is a string $fst = (string) $foo; // $fst is also a string  // This prints out that "they are the same" if ($fst === $str) {     echo "they are the same"; } ?> |   当在某些类型之间强制转换时确切地会发生什么可能不是很明显。更多信息见如下小节：   * [转换为布尔值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html#language.types.boolean.casting) * [转换为整型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html#language.types.integer.casting) * [转换为浮点型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html#language.types.float.casting) * [转换为字符串](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html#language.types.string.casting) * [转换为数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html#language.types.array.casting) * [转换为对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html#language.types.object.casting) * [转换为资源](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.resource.html#language.types.resource.casting) * [类型比较表](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/types.comparisons.html) |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.pseudo-types.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | | 本文档中使用的伪类型 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html) | 变量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html) | | |  | | | |
|  | **章 7. 变量**  **目录**  [基础](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html#language.variables.basics)  [预定义变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html)  [变量范围](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html)  [可变变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html)  [PHP 的外部变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.external.html)  **基础**  PHP 中一个美元符号后面跟上一个变量名称，即表示一个变量。变量的名称是对大小写敏感的。  变量名与 PHP 中其它的标签一样遵循相同的规则。一个有效的变量名由字母或者下划线开头，后面跟上任意数量的字母，数字，或者下划线。按照正常的正则表达式，它将被表述为：'[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*'  **注:** 字母为 a-z，A-Z，ASCII 字符从 127 到 255（0x7f-0xff）。   |  | | --- | | <?php $var = "Bob"; $Var = "Joe"; echo "$var, $Var";      // outputs "Bob, Joe"  $4site = 'not yet';     // invalid; starts with a number $\_4site = 'not yet';    // valid; starts with an underscore $t?yte = 'mansikka';    // valid; '洄 is (Extended) ASCII 228. ?> |   PHP 3 中，变量总是传值赋值。那也就是说，当你将一个表达式的值赋予一个变量时，整个原始表达式的值被赋值到目标变量。这意味着，例如，当一个变量的值赋予另外一个变量时，改变其中一个变量的值，将不会影响到另外一个变量。有关这种类型的赋值操作，请参阅[表达式](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html)一章。  PHP 4 提供了另外一种方式给变量赋值：[传地址赋值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html)。这意味着新的变量简单的引用（换言之，“成为其别名” 或者 “指向”）了原始变量。改动新的变量将影响到原始变量，反之亦然。这同样意味着其中没有执行复制操作；因而，这种赋值操作更加快速。尽管如此，任何提速的操作只有在紧密循环或者大[数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html)或者[对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.object.html)才可能被注意到。  使用传地址赋值，简单地追加一个（&）符号到将要赋值的变量前（源变量）。例如，下列代码片断两次输出‘My name is Bob’：   |  | | --- | | <?php $foo = 'Bob';              // Assign the value 'Bob' to $foo $bar = &$foo;              // Reference $foo via $bar. $bar = "My name is $bar";  // Alter $bar... echo $bar; echo $foo;                 // $foo is altered too. ?> |   需要注意的是只有命名变量才可以传地址赋值，这一点非常重要。   |  | | --- | | <?php $foo = 25; $bar = &$foo;      // This is a valid assignment. $bar = &(24 \* 7);  // Invalid; references an unnamed expression.  function test() {    return 25; }  $bar = &test();    // Invalid. ?> | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html) | | 类型戏法 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 预定义变量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | 章 7. 变量 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html) | | |  | | | |
|  | **预定义变量**  PHP 提供了大量的预定义变量。由于许多变量依赖于运行的服务器的版本和设置，及其它因素，所以并没有详细的说明文档。一些预定义变量在 PHP 以[命令行](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.commandline.html)形式运行时并不生效。有关这些变量的详细列表，请参阅“[保留的预定义变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html)”一章。   |  | | --- | | **警告** | | PHP 4.2.0 以及后续版本中，PHP 指令 [register\_globals](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.register-globals) 的默认值为 *off*。这是 PHP 的一个主要变化。让 register\_globals 的值为 *off* 将影响到预定义变量集在全局范围内的有效性。例如，为了得到 DOCUMENT\_ROOT 的值，你将必须使用 $\_SERVER['DOCUMENT\_ROOT'] 代替 $DOCUMENT\_ROOT，又如，使用 $\_GET['id'] 来代替 $id 从 URL http://www.example.com/test.php?id=3 中获取 id 值，亦或使用 $\_ENV['HOME'] 来代替 $HOME 获取环境变量 HOME 的值。  更多相关信息，请阅读配置项目 [register\_globals](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.register-globals)，有关安全性的一章[使用 Register Globals](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/security.registerglobals.html)，以及 PHP [4.1.0](http://www.php.net/release_4_1_0.php" \t "_top) 和 [4.2.0](http://www.php.net/release_4_2_0.php" \t "_top) 的发行通告.  请优先使用可用的 PHP 预定义变量，如 [超级全局数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html#language.variables.superglobals)。 |   从 PHP 4.1.0 开始，PHP 提供了一套附加的预定数组，这些数组变量包含了来自 Web 服务器（如果可用），运行环境，和用户输入的数据。这些数组非常特别，它们在全局范围内自动生效，例如，在任何范围内自动生效。为此，它们常因是 "autoglobals" 或者 "superglobals" 而闻名。（PHP 中尚且没有一种可使用户自定义超级全局变量的机制）超级全局变量罗列于下文中；但是为了得到它们的内容和关于 PHP 预定义变量的进一步的讨论以及它们的本质，请参阅[预定义变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html)。而且，你也将注意到旧的预定义数组（$HTTP\_\*\_VARS）仍旧存在。 在 PHP 5.0.0 中，长的 PHP [预定义](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html)数组可以通过设置 [register\_long\_arrays](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.register-long-arrays) 来屏蔽。  **可变变量:** 超级全局变量不能被用作[可变变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html).  如果某些 [variables\_order](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.variables-order) 中的变量没有设定，它们的对应的 PHP 预定义数组也是空的。  **PHP 超全局变量**  [$GLOBALS](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.globals)  包含一个引用指向每个当前脚本的全局范围内有效的变量。该数组的键标为全局变量的 名称。从 PHP 3 开始存在 $GLOBALS 数组。  [$\_SERVER](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.server)  变量由 Web 服务器设定或者直接与当前脚本的执行环境相关联。类似于旧数组 $HTTP\_SERVER\_VARS 数组（依然有效，但反对使用）。  [$\_GET](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.get)  经由 HTTP GET 方法提交至脚本的变量。类似于旧数组 $HTTP\_GET\_VARS 数组（依然有效，但反对使用）。  [$\_POST](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.post)  经由 HTTP POST 方法提交至脚本的变量。类似于旧数组 $HTTP\_POST\_VARS 数组（依然有效，但反对使用）。  [$\_COOKIE](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.cookies)  经由 HTTP Cookies 方法提交至脚本的变量。类似于旧数组 $HTTP\_COOKIE\_VARS 数组（依然有效，但反对使用）。  [$\_FILES](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.files)  经由 HTTP POST 文件上传而提交至脚本的变量。类似于旧数组 $HTTP\_POST\_FILES 数组（依然有效，但反对使用）。详细信息请参阅 [POST 方法上传](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.file-upload.html#features.file-upload.post-method)。  [$\_ENV](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.environment)  执行环境提交至脚本的变量。类似于旧数组 $HTTP\_ENV\_VARS 数组（依然有效，但反对使用）。  [$\_REQUEST](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.request)  经由 GET，POST 和 COOKIE 机制提交至脚本的变量，因此该数组并不值得信任。所有包含在该数组中的变量的存在与否以及变量的顺序均按照 php.ini 中的 [variables\_order](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.variables-order) 配置指示来定义。该数组没有直接模拟 PHP 4.1.0 的早期版本。参见 [**import\_request\_variables()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.import-request-variables.html)。   |  | | --- | | **注意** | | 自 PHP 4.3.0 起，$\_FILES 中的文件信息不再存在于 $\_REQUEST 中。 |   **注:** 当运行于[命令行](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.commandline.html)模式时，这个数组将*不会*包含 argv 和 argc 入口；它们已经存在于数组 $\_SERVER 中。  [$\_SESSION](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.session)  当前注册给脚本会话的变量。类似于旧数组 $HTTP\_SESSION\_VARS 数组（依然有效，但反对使用）。详细信息，请参照 [Session 处理函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.session.html)章节。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html) | | 变量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | 变量范围 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html) | 章 7. 变量 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html) | | |  | | | |
|  | 变量范围 变量的范围即它定义的上下文背景（译者:说白了，也就是它的生效范围）。大部分的 PHP 变量只有一个单独的范围。这个单独的范围跨度同样包含了 include 和 require 引入的文件。范例:   |  | | --- | | <?php $a = 1; include "b.inc"; ?> |   这里变量 $a 将会在包含文件 b.inc 中生效。但是，在用户自定义函数中，一个局部函数范围将被引入。任何用于函数内部的变量按缺省情况将被限制在局部函数范围内。范例：   |  | | --- | | <?php $a = 1; /\* global scope \*/  function Test() {     echo $a; /\* reference to local scope variable \*/ }  Test(); ?> |   这个脚本不会有任何输出，因为 echo 语句引用了一个局部版本的变量 $a，而且在这个范围内，它并没有被赋值。你可能注意到 PHP 的全局变量和 C 语言有一点点不同，在 C 语言中，全局变量在函数中自动生效，除非被局部变量覆盖。这可能引起一些问题，有些人可能漫不经心的改变一个全局变量。PHP 中全局变量在函数中使用时必须申明为全局。 The global keyword 首先，一个使用 global 的例子：   |  |  | | --- | --- | | **例子 7-1. 使用 global**   |  | | --- | | <?php $a = 1; $b = 2;  function Sum() {     global $a, $b;      $b = $a + $b; }  Sum(); echo $b; ?> | |   以上脚本的输出将是 "3"。在函数中申明了全局变量 $a 和 $b，任何变量的所有引用变量都会指向到全局变量。对于一个函数能够申明的全局变量的最大个数，PHP 没有限制。  在全局范围内访问变量的第二个办法，是用特殊的 PHP 自定义 $GLOBALS 数组。前面的例子可以写成：   |  |  | | --- | --- | | **例子 7-2. 使用 $GLOBALS 替代 global**   |  | | --- | | <?php $a = 1; $b = 2;  function Sum() {     $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"]; }  Sum(); echo $b; ?> | |   在 $GLOBALS 数组中，每一个变量为一个元素，键名对应变量名，值变量的内容。$GLOBALS 之所以在全局范围内存在，是因为 $GLOBALS 是一个[超全局变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html#language.variables.superglobals)。以下范例显示了超全局变量的用处：   |  |  | | --- | --- | | **例子 7-3. 演示超全局变量和作用域的例子**   |  | | --- | | <?php function test\_global() {     // 大多数的预定义变量并不 "super"，它们需要用 'global' 关键字来使它们在函数的本地区域中有效。     global $HTTP\_POST\_VARS;      print $HTTP\_POST\_VARS['name'];      // Superglobals 在任何范围内都有效，它们并不需要 'global' 声明。Superglobals 是在 PHP 4.1.0 引入的。     print $\_POST['name']; } ?> | |  使用静态变量 变量范围的另一个重要特性是*静态变量*（static variable）。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。看看下面的例子：   |  |  | | --- | --- | | **例子 7-4. 演示需要静态变量的例子**   |  | | --- | | <?php function Test () {     $a = 0;     echo $a;     $a++; } ?> | |   本函数没什么用处，因为每次调用时都会将 $a 的值设为 0 并输出 "0"。将变量加一的 $a++ 没有作用，因为一旦退出本函数则变量 $a 就不存在了。要写一个不会丢失本次计数值的计数函数，要将变量 $a 定义为静态的：   |  |  | | --- | --- | | **例子 7-5. 使用静态变量的例子**   |  | | --- | | <?php function Test() {     static $a = 0;     echo $a;     $a++; } ?> | |   现在，每次调用 Test() 函数都会输出 $a 的值并加一。  静态变量也提供了一种处理递归函数的方法。递归函数是一种调用自己的函数。写递归函数时要小心，因为可能会无穷递归下去。必须确保有充分的方法来中止递归。一下这个简单的函数递归计数到 10，使用静态变量 $count 来判断何时停止：   |  |  | | --- | --- | | **例子 7-6. 静态变量与递归函数**   |  | | --- | | <?php function Test() {     static $count = 0;      $count++;     echo $count;     if ($count < 10) {         Test ();     }     $count--; } ?> | |   **注:** 静态变量可以按照上面的例子声明。如果在声明中用表达式的结果对其赋值会导致解析错误。   |  |  | | --- | --- | | **例子 7-7. 声明静态变量**   |  | | --- | | <?php function foo(){     static $int = 0;          // correct     static $int = 1+2;        // wrong  (as it is an expression)     static $int = sqrt(121);  // wrong  (as it is an expression too)      $int++;     echo $int; } ?> | |  全局和静态变量的引用 在 Zend 引擎 1 代，驱动了 PHP4，对于变量的 [static](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html#language.variables.scope.static) 和 [global](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html#language.variables.scope.global) 定义是以 [references](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) 的方式实现的。例如，在一个函数域内部用 global 语句导入的一个真正的全局变量实际上是建立了一个到全局变量的引用。这有可能导致预料之外的行为，如以下例子所演示的：   |  | | --- | | <?php function test\_global\_ref() {     global $obj;     $obj = &new stdclass; }  function test\_global\_noref() {     global $obj;     $obj = new stdclass; }  test\_global\_ref(); var\_dump($obj); test\_global\_noref(); var\_dump($obj); ?> |   执行以上例子会导致如下输出：   |  | | --- | | NULL  object(stdClass)(0) {  } |   类似的行为也适用于 static 语句。引用并不是静态地存储的：   |  | | --- | | <?php function &get\_instance\_ref() {     static $obj;      echo "Static object: ";     var\_dump($obj);     if (!isset($obj)) {         // 将一个引用赋值给静态变量         $obj = &new stdclass;     }     $obj->property++;     return $obj; }  function &get\_instance\_noref() {     static $obj;      echo "Static object: ";     var\_dump($obj);     if (!isset($obj)) {         // 将一个对象赋值给静态变量         $obj = new stdclass;     }     $obj->property++;     return $obj; }  $obj1 = get\_instance\_ref(); $still\_obj1 = get\_instance\_ref(); echo "\n"; $obj2 = get\_instance\_noref(); $still\_obj2 = get\_instance\_noref(); ?> |   执行以上例子会导致如下输出：   |  | | --- | | Static object: NULL  Static object: NULL  Static object: NULL  Static object: object(stdClass)(1) {  ["property"]=>  int(1)  } |   上例演示了当把一个引用赋值给一个静态变量时，第二次调用 &get\_instance\_ref() 函数时其值并没有被*记住*。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html) | | 预定义变量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | 可变变量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html) | 章 7. 变量 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.external.html) | | |  | | | |
|  | **可变变量**  有时候使用可变变量名是很方便的。就是说，一个变量的变量名可以动态的设置和使用。一个普通的变量通过声明来设置，例如：   |  | | --- | | <?php $a = "hello"; ?> |   一个可变变量获取了一个普通变量的值作为这个可变变量的变量名。在上面的例子中 *hello* 使用了两个美元符号（$）以后，就可以作为一个可变变量的变量了。例如：   |  | | --- | | <?php $$a = "world"; ?> |   这时，两个变量都被定义了：$a 的内容是“hello”并且 $hello 的内容是“world”。因此，可以表述为：   |  | | --- | | <?php echo "$a ${$a}"; ?> |   以下写法更准确并且会输出同样的结果：   |  | | --- | | <?php echo "$a $hello"; ?> |   它们都会输出：hello world。  要将可变变量用于数组，必须解决一个模棱两可的问题。这就是当写下 $$a[1] 时，解析器需要知道是想要 $a[1] 作为一个变量呢，还是想要 $$a 作为一个变量并取出该变量中索引为 [1] 的值。解决此问题的语法是，对第一种情况用 ${$a[1]}，对第二种情况用 ${$a}[1]。  注意可变变量不能用于 PHP 的[超全局变量数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html#language.variables.superglobals)。这意味着不能这样用：${$\_GET}。 如果想要一种处理超全局变量和老的 HTTP\_\*\_VARS 的方法，应该尝试[引用](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html)它们。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.external.html) | | 变量范围 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | PHP 的外部变量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html) | 章 7. 变量 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) | | |  | | | |
|  | PHP 的外部变量HTML 表单（GET 和 POST） 当一个表单体交给 PHP 脚本时，表单中的信息会自动在脚本中可用。有很多方法访问此信息，例如：   |  |  | | --- | --- | | **例子 7-8. 一个简单的 HTML 表单**   |  | | --- | | <form action="foo.php" method="POST">  Name: <input type="text" name="username"><br>  Email: <input type="text" name="email"><br>  <input type="submit" name="submit" value="Submit me!">  </form> | |   根据特定的设置和个人的喜好，有很多种方法访问 HTML 表单中的数据。例如：   |  |  | | --- | --- | | **例子 7-9. 从一个简单的 POST HTML 表单访问数据**   |  | | --- | | <?php  // 自 PHP 4.1.0 起可用  print $\_POST['username'];  print $\_REQUEST['username'];  import\_request\_variables('p', 'p\_');  print $p\_username;  // 自 PHP 3 起可用。自 PHP 5.0.0 起，这些较长的预定义变量  // 可用 register\_long\_arrays 指令关闭。  print $HTTP\_POST\_VARS['username'];  // 如果 PHP 指令 register\_globals = on 时可用。不过自  // PHP 4.2.0 起默认值为 register\_globals = off。  // 不提倡使用/依赖此种方法。  print $username;  ?> | |   使用 GET 表单也类似，只不过要用适当的 GET 预定义变量。GET 也适用于 QUERY\_STRING（URL 中在“?”之后的信息）。因此，举例说，http://www.example.com/test.php?id=3 包含有可用 $\_GET['id'] 访问的 GET 数据。参见 [$\_REQUEST](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.request) 和 [**import\_request\_variables()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.import-request-variables.html)。  **注:** [超全局变量数组](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html#language.variables.superglobals)，和 $\_POST 以及 $\_GET 一样，自 PHP 4.1.0 起可用。  如同所示，在 PHP 4.2.0 之前 [register\_globals](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.register-globals) 的默认值是 *on*。在 PHP 3 中其值总是 on。PHP 社区鼓励大家不要依赖此指令，建议在编码时假定其为 *off*。  **注:** [magic\_quotes\_gpc](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.info.html#ini.magic-quotes-gpc) 配置指令影响到 Get，Post 和 Cookie 的值。如果打开，值 (It's "PHP!") 会自动转换成 (It\'s \"PHP!\")。数据库的插入就需要转义。参见 [**addslashes()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.addslashes.html)，[**stripslashes()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.stripslashes.html) 和 [magic\_quotes\_sybase](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.sybase.html#ini.magic-quotes-sybase)。  PHP 也懂得表单变量上下文中的数组（参见[相关常见问题](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/faq.html.html)）。例如可以将相关的变量编成组，或者用此特性从多选输入框中取得值。例如，将一个表单 POST 给自己并在提交时显示数据：   |  |  | | --- | --- | | **例子 7-10. 更复杂的表单变量**   |  | | --- | | <?php if (isset($\_POST['action']) && $\_POST['action'] == 'submitted') {     print '<pre>';      print\_r($\_POST);     print '<a href="'. $\_SERVER['PHP\_SELF'] .'">Please try again</a>';      print '</pre>'; } else { ?> <form action="<?php echo $\_SERVER['PHP\_SELF']; ?>" method="POST">     Name:  <input type="text" name="personal[name]"><br>     Email: <input type="text" name="personal[email]"><br>     Beer: <br>     <select multiple name="beer[]">         <option value="warthog">Warthog</option>         <option value="guinness">Guinness</option>         <option value="stuttgarter">Stuttgarter Schwabenbr漉</option>     </select><br>     <input type="hidden" name="action" value="submitted">     <input type="submit" name="submit" value="submit me!"> </form> <?php } ?> | |   在 PHP 3 中，变量使用中的数组仅限于一维数组。在 PHP 4 中，没有此种限制。 IMAGE SUBMIT 变量名 当提交表单时，可以用一幅图像代替标准的提交按钮，用类似这样的标记：   |  | | --- | | <input type="image" src="image.gif" name="sub"> |   当用户点击到图像中的某处时，相应的表单会被传送到服务器，并加上两个变量 sub\_x 和 sub\_y。它们包含了用户点击图像的坐标。有经验的用户可能会注意到被浏览器发送的实际变量名包含的是一个点而不是下划线，但 PHP 自动将点转换成了下划线。 HTTP Cookies PHP 透明地支持 [Netscape 规范](http://www.netscape.com/newsref/std/cookie_spec.html" \t "_top)定义中的 HTTP cookies。Cookies 是一种在远端浏览器端存储数据并能追踪或识别再次访问的用户的机制。可以用 [**setcookie()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.setcookie.html) 函数设定 cookies。Cookies 是 HTTP 信息头中的一部分，因此 SetCookie 函数必须在向浏览器发送任何输出之前调用。对于 [**header()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.header.html) 函数也有同样的限制。Cookie 数据会在相应的 cookie 数据数组中可用，例如 $\_COOKIE，$HTTP\_COOKIE\_VARS 和 $\_REQUEST。更多细节和例子见 [**setcookie()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.setcookie.html) 手册页面。  如果要将多个值赋给一个 cookie 变量，必须将其赋成数组。例如：   |  | | --- | | <?php   setcookie("MyCookie[foo]", "Testing 1", time()+3600);   setcookie("MyCookie[bar]", "Testing 2", time()+3600); ?> |   这将会建立两个单独的 cookie，尽管 MyCookie 在脚本中是一个单一的数组。如果想在仅仅一个 cookie 中设定多个值，考虑先在值上使用 [**serialize()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.serialize.html) 或 [**explode()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.explode.html)。  注意在浏览器中一个 cookie 会替换掉上一个同名的 cookie，除非路径或者域不同。因此对于购物框程序可以保留一个计数器并一起传递，例如：   |  |  | | --- | --- | | **例子 7-11. 一个** [**setcookie()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.setcookie.html) **的示例**   |  | | --- | | <?php if (isset($\_COOKIE['count'])) {     $count = $\_COOKIE['count'] + 1; } else {     $count = 1; } setcookie("count", $count, time()+3600); setcookie("Cart[$count]", $item, time()+3600); ?> | |  变量名中的点 通常，PHP 不会改变传递给脚本中的变量名。然而应该注意到点（dot，period，full stop）不是 PHP 变量名中的合法字符。至于原因，看看：   |  | | --- | | <?php $varname.ext;  /\* 非法变量名 \*/ ?> |   这时，解析器看到是一个名为 $varname 的变量，后面跟着一个字符串连接运算符，后面跟着一个裸字符串（例如没有加引号的字符串，且不匹配任何已知的健名或保留字）'ext'。很明显这不是想要的结果。  出于此原因，要注意 PHP 将会自动将变量名中的点替换成下划线。 确定变量类型 因为 PHP 会判断变量类型并在需要时进行转换（通常情况下），因此在某一时刻给定的变量是何种类型并不明显。PHP 包括几个函数可以判断变量的类型，例如：[**gettype()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.gettype.html)，[**is\_array()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-array.html)，[**is\_float()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-float.html)，[**is\_int()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-int.html)，[**is\_object()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-object.html) 和 [**is\_string()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.is-string.html)。参见[类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html)一章。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) | | 可变变量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.html) | 常量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.external.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html) | | |  | | | |
|  | **章 8. 常量**  **目录**  [语法](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html#language.constants.syntax)  [预定义常量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html)  常量是一个简单值的标识符（名字）。如同其名称所暗示的，在脚本执行期间该值不能改变（除了所谓的[魔术常量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html)，它们其实不是常量）。常量默认为大小写敏感。按照惯例常量标识符总是大写的。  常量名和其它任何 PHP 标签遵循同样的命名规则。合法的常量名以字母或下划线开始，后面跟着任何字母，数字或下划线。用正则表达式是这样表达的：[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*  **注:** 在这里，字母是 a-z，A-Z，以及从 127 到 255（0x7f-0xff）的 ASCII 字符。  和 [superglobals](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.predefined.html) 一样，常量的范围是全局的。不用管作用域就可以在脚本的任何地方访问常量。有关作用得更多信息请阅读手册中的[变量范围](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.scope.html)。  **语法**  可以用 [**define()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.define.html) 函数来定义常量。一个常量一旦被定义，就不能再改变或者取消定义。  常量只能包含标量数据（[**boolean**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.boolean.html)，[**integer**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.integer.html)，[**float**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.float.html) 和 [**string**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)）。  可以简单的通过指定其名字来取得常量的值，*不要*在常量前面加上 $ 符号。如果常量名是动态的，也可以用函数 [**constant()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.constant.html) 来读取常量的值。用 [**get\_defined\_constants()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.get-defined-constants.html) 可以获得所有已定义的常量列表。  **注:** 常量和（全局）变量在不同的名字空间中。这意味着例如 **TRUE** 和 $TRUE 是不同的。  如果使用了一个未定义的常量，PHP 假定你想要的是该常量本身的名字，如同你用字符串调用它一样（CONSTANT 对应 "CONSTANT"）。此时将发出一个 [E\_NOTICE](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.errorfunc.html) 级的错误。参见手册中为什么 [$foo[bar]](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html#language.types.array.foo-bar) 是错误的（除非你事先用 [**define()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.define.html) 将 bar 定义为一个常量）。如果你只想检查是否定义了某常量，用 [**defined()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.defined.html) 函数。  常量和变量不同：   * 常量前面没有美元符号（$）； * 常量只能用 [**define()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.define.html) 函数定义，而不能通过赋值语句； * 常量可以不用理会变量范围的规则而在任何地方定义和访问； * 常量一旦定义就不能被重新定义或者取消定义； * 常量的值只能是标量。  |  |  | | --- | --- | | **例子 8-1. 定义常量**   |  | | --- | | <?php define("CONSTANT", "Hello world."); echo CONSTANT; // outputs "Hello world." echo Constant; // outputs "Constant" and issues a notice. ?> | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.external.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html) | | PHP 的外部变量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 预定义常量 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) | 章 8. 常量 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html) | | |  | | | |
|  | **预定义常量**  PHP 向它运行的任何脚本提供了大量的预定义常量。不过很多常量都是由不同的扩展库定义的，只有在加载了这些扩展库时才会出现，或者动态加载后，或者在编译时已经包括进去了。  有四个魔术常量根据它们使用的位置而改变。例如 **\_\_LINE\_\_** 的值就依赖于它在脚本中所处的行来决定。这些特殊的常量不区分大小写，如下：  **表格 8-1. 几个 PHP 的“魔术常量”**   | **名称** | **说明** | | --- | --- | | **\_\_LINE\_\_** | 文件中的当前行号。 | | **\_\_FILE\_\_** | 文件的完整路径和文件名。 | | **\_\_FUNCTION\_\_** | 函数名称（这是 PHP 4.3.0 新加的）。 | | **\_\_CLASS\_\_** | 类的名称（这是 PHP 4.3.0 新加的）。 | | **\_\_METHOD\_\_** | 类的方法名（这是 PHP 5.0.0 新加的）。 |   预定义常量的列表见[预定义常量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.constants.html)一节。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html) | | 常量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.html) | 表达式 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | | |  | | | |
|  | **章 9. 表达式**  表达式是 PHP 最重要的基石。在 PHP 中，几乎你所写的任何东西都是一个表达式。简单但却最精确的定义一个表达式的方式就是“anything that has a value”。  最基本的表达式形式是常量和变量。当你键入“$a = 5”，即将值 '5' 分配给变量 $a。'5'，很明显，值为 5，换句话说 '5' 是一个值为5的表达式（既然如此，'5' 是一个整型常量）。  赋值之后，你所盼望的情况是 $a 的值为5，因而如果你写下 $b = $a，期望的是它犹如 $b = 5 一样。换句话说，$a 是一个值也为 5 的表达式。如果一切运行正确，那这正是将要发生的正确结果。  稍微复杂的表达式例子就是函数。例如，考虑下面的函数：   |  | | --- | | <?php function foo () {     return 5; } ?> |   假定你已经熟悉了函数的概念（如果不是的话，请看一下函数的相关章节），那么键入 $c = foo() 从本质上来说就如写下 $c = 5，而且你是正确的。 函数也是表达式，表达式的值即为它们的返回值。既然 foo() 返回5，表达式 ‘foo()’ 的值也是5。通常函数不会仅仅返回一个静态值，而可能会计算一些东西。  当然，PHP 中的值常常并非是整型的。PHP 支持三种标量值类型：整型值，浮点值和字符串值（标量值不能拆分为更小的单元，比如：数组）。PHP 也支持两种复合类型：数组和对象。这两种类型具可以赋值给变量或者从函数返回。  到目前为止，PHP/FI 2 的用户不应该感到任何的变化。然而，当许多其它语言为之努力的时候，PHP 在相同道路上促进了表达式的成长。PHP 是一种面向表达式的语言，从这一方面来讲几乎一切都是表达式。考虑刚才我们已经研究过的例子，“$a = 5”。可以轻松的看到这里有两个相关的值，整型常量 5，而且变量 $a 的值也被更新为 5。但是事实是：这里只有一个相关的附加值，即被分配的值本身。赋值操作计算需分配的值，即 5。实际上，意味着“$a = 5”，不必管它是做什么的，是一个值为 5 的表达式。因而，一些像这样的代码“$b = ($a = 5)”和“$a =5; $b=5”（分号标志着语句的结束）。因为赋值操作的顺序是由右到左的，你也可以这么写“$b = $a =5”。  另外一个很好的面向表达式的例子就是前、后递增和递减。PHP/FI 2 和多数其它语言的用户应该比较熟悉变量 ++ 和变量 -- 符号。即递增和递减操作符。在 PHP/FI 2 中，语句“$a++”没有值（不是表达式），这样的话你便不能为其赋值或者以任何其它方式来使用它。PHP 通过将其变为了表达式，类似 C 语言，增强了递增/递减的能力。在 PHP 和 C 语言 中，有两种类型的递增前递增和后递增，本质上来讲，前递增和后递增均增加了变量的值，并且对于变量的影响是相同的。不同的是递增表达式的值。前递增，写做“++$variable”，求增加后的值（PHP 在读取变量的值之前，增加变量的值，因而称之为“前递增”）。后递增，写做‘$variable++’，求变量未递增之前的原始值（PHP 在读取变量的值之后，增加变量的值，因而叫做‘后递增’）。【译者注：前递增，++$a，则该表达式的值加 1；后递增，$a++，则该表达式的值不变。】  一个常用到表达式类型是比较表达式。这些表达式求值 0 或者 1，即 **FALSE** 或者 **TRUE**(分别的)。PHP 支持 >（大于），>=（大于等于），==（等于），!=（不等于），<（小于），<= (小于等于)。这些表达式都是在条件判断语句，比如，if 语句中最常用的。  这里，我们将要研究的最后一个例子是组合的赋值操作符表达式。你已经知道如果想要为变量 $a 加1，可以简单的写‘$a++’或者‘++$a’。但是如果你想为变量增加大于 1 的值，比如 3，该怎么做？你可以多次写‘$a++’，但这样明显不是一种高效舒适的方法，一个更加通用的做法是‘$a = $a + 3’。‘$a + 3’ 计算 $a 加上 3 的值，并且得到的值重新赋予变量 $a，于是 $a 的值增加了3。在 PHP 及其它几种类似 C 的语言中，你可以以一种更加简短的形式完成上述功能，因而也更加清楚快捷。为 $a 的当前值加 3，可以这样写：$a += 3。这里的意思是“取变量 $a 的值，加 3，得到的结果再次分配给变量 $a”。除了更简略和清楚外，也可以更快的运行。‘$a += 3’的值，如同一个正常赋值操作的值，是赋值后的值。注意它不是 3，而是 $a 加上3的组合值（即已经分配给 $a 的值）【译者注：这里表达式‘$a += 3’的值是什么呢？不是 3，也不是 $a 的原始值，而是完成 +3 操作后变量 $a 的值】。 任何两位操作符都可以使用在复制操作符模式，例如‘$a -= 5’（从变量 $a 的值中减去 5），‘$b \*= 7’ （变量 $b 乘以 7），等等。  有一些表达式，如果你没有在别的语言中看到过的话，可能认为它们是多余的，如三重操作符：   |  | | --- | | $first ? $second : $third |   如果第一个子表达式的值是 **TRUE**（非零），那么计算第二个子表达式的值，其值即为整个表达式的值。否则，将是第三个子表达式的值。  下面的例子一般来说应该可以稍微帮你理解前、后递增和表达式：   |  | | --- | | <?php function double($i) {     return $i\*2; } $b = $a = 5;        /\* assign the value five into the variable $a and $b \*/ $c = $a++;          /\* post-increment, assign original value of $a                        (5) to $c \*/ $e = $d = ++$b;     /\* pre-increment, assign the incremented value of                        $b (6) to $d and $e \*/  /\* at this point, both $d and $e are equal to 6 \*/  $f = double($d++);  /\* assign twice the value of $d before                        the increment, 2\*6 = 12 to $f \*/ $g = double(++$e);  /\* assign twice the value of $e after                        the increment, 2\*7 = 14 to $g \*/ $h = $g += 10;      /\* first, $g is incremented by 10 and ends with the                        value of 24. the value of the assignment (24) is                        then assigned into $h, and $h ends with the value                        of 24 as well. \*/ ?> |   在本章的开始，我们说过我们将会描述多种语句类型，并且如同许诺的那样，表达式可以是语句。尽管如此，不是每个表达式都是一个语句。而这样的话，一个语句，它的形式是 'expr' ';'，一个表达式有一个分号结尾。在‘$b=$a=5;’，$a = 5 是一个有效的表达式，但是它却不是一个语句。‘$b=$a=5;’ 却是一个有效的语句。  最后一件值得提起的事情就是表达式的真实值。在许多事件中，大体上主要是在条件执行和循环中，不要专注于表达式中明确的值，反而要注意表达式的值是否是 **TRUE** 或者 **FALSE**。常量 **TRUE** 和 **FALSE** （大小写无关）是两种可能的 Boolean 值。如果有必要，一个表达式将自动转换为 Boolean。参见[类型强制转换](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.type-juggling.html#language.types.typecasting)一节。  PHP 提供了一套完整强大的表达式，而为它提供完整的文件资料已经超出了本手册的范围。上面的例子应该为你提供了一个好的关于什么是表达式和怎样构建一个有用的表达式的概念。在本手册的其余部分，我们将始终使用 *expr*来表示一个有效的 PHP 表达式。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.constants.predefined.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | | 预定义常量 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.arithmetic.html) | | |  | | | |
|  | **章 10. 运算符**  **目录**  [运算符优先级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html#language.operators.precedence)  [算术运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.arithmetic.html)  [赋值运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html)  [位运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.bitwise.html)  [比较运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.comparison.html)  [错误控制运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.errorcontrol.html)  [执行运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.execution.html)  [加一／减一运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.increment.html)  [逻辑运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.logical.html)  [字符串运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html)  [数组运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.array.html)  **运算符优先级**  运算符优先级指定了两个表达式绑定得有多“紧密”。例如，表达式 1 + 5 \* 3 的结果是 16 而不是 18 是因为乘号（“\*”）的优先级比加号（“+”）高。必要时可以用括号来强制改变优先级。例如：(1 + 5) \* 3 的值为 18。  下表从低到高列出了运算符的优先级。  **表格 10-1. 运算符优先级**   | **结合方向** | **运算符** | | --- | --- | | 左 | , | | 左 | or | | 左 | xor | | 左 | and | | 右 | print | | 右 | = += -= \*= /= .= %= &= |= ^= ~= <<= >>= | | 左 | ? : | | 左 | || | | 左 | && | | 左 | | | | 左 | ^ | | 左 | & | | 无 | == != === !== | | 无 | < <= > >= | | 左 | << >> | | 左 | + - . | | 左 | \* / % | | 右 | ! ~ ++ -- (int) (float) (string) (array) (object) @ | | 右 | [ | | 无 | new |   **注:** 尽管 ! 比 = 的优先级高，PHP 仍旧允许类似如下的表达式：if (!$a = foo())，在此例中 foo() 的输出被赋给了 $a。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.arithmetic.html) | | 表达式 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 算术运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html) | | |  | | | |
|  | **算术运算符**  还记得学校里学到的基本数学知识吗？就和它们一样。  **表格 10-2. 算术运算符**   | **例子** | **名称** | **结果** | | --- | --- | --- | | $a + $b | 加法 | $a 和 $b 的和。 | | $a - $b | 减法 | $a 和 $b 的差。 | | $a \* $b | 乘法 | $a 和 $b 的积。 | | $a / $b | 除法 | $a 除以 $b 的商。 | | $a % $b | 取模 | $a 除以 $b 的余数。 |   除号（“/”）总是返回浮点数，即使两个运算数是整数（或由字符串转换成的整数）也是这样。  请查阅手册“[数学函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.math.html)”有关章节。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html) | | 运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 赋值运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.arithmetic.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.bitwise.html) | | |  | | | |
|  | **赋值运算符**  基本的赋值运算符是“=”。你一开始可能会以为它是“等于”，其实不是的。它实际上意味着把右边表达式的值赋给左运算数。  赋值运算表达式的值也就是所赋的值。也就是说，“$a = 3”的值是 3。这样就可以使你做一些小技巧：   |  | | --- | | $a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4. |   在基本赋值运算符之外，还有适合于所有二元算术和字符串运算符的“组和运算符”，这可以让你在一个表达式中使用它的值并把表达式的结果赋给它，例如：   |  | | --- | | $a = 3; $a += 5; // sets $a to 8, as if we had said: $a = $a + 5; $b = "Hello "; $b .= "There!"; // sets $b to "Hello There!", just like $b = $b . "There!"; |   注意赋值运算将原变量的值拷贝到新变量中（传值赋值），所以改变其中一个并不影响另一个。这也适合于你在在紧密循环中拷贝一些值例如大数值。PHP 4 支持引用赋值，用 $var = &$othervar; 语法，但在 PHP 3 中不可能这样做。“引用赋值”意味着两个变量都指向同一个数据，没有任何数据的拷贝。有关引用的更多信息见[引用的说明](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html)。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.arithmetic.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.bitwise.html) | | 算术运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 位运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.comparison.html) | | |  | | | |
|  | **位运算符**  位运算符允许对整型数中指定的位进行置位。如果左右参数都是字符串，则位运算符将操作这个字符串中的字符。   |  | | --- | | <?php     echo 12 ^ 9; // Outputs '5'      echo "12" ^ "9"; // Outputs the Backspace character (ascii 8)                      // ('1' (ascii 49)) ^ ('9' (ascii 57)) = #8      echo "hallo" ^ "hello"; // Outputs the ascii values #0 #4 #0 #0 #0                             // 'a' ^ 'e' = #4 ?> |   **表格 10-3. 位运算符**   | **例子** | **名称** | **结果** | | --- | --- | --- | | $a & $b | And（按位与） | 将在 $a 和 $b 中都为 1 的位设为 1。 | | $a | $b | Or（按位或） | 将在 $a 或者 $b 中为 1 的位设为 1。 | | $a ^ $b | Xor（按位异或） | 将在 $a 和 $b 中不同的位设为 1。 | | ~ $a | Not（按位非） | 将 $a 中为 0 的位设为 1，反之亦然。 | | $a << $b | Shift left（左移） | 将 $a 中的位向左移动 $b 次（每一次移动都表示“乘以 2”）。 | | $a >> $b | Shift right（右移） | 将 $a 中的位向右移动 $b 次（每一次移动都表示“除以 2”）。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.comparison.html) | | 赋值运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 比较运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.bitwise.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.errorcontrol.html) | | |  | | | |
|  | **比较运算符**  比较运算符，如同它们名称所暗示的，允许你对两个值进行比较。你还可以参考 [PHP 类型比较表](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/types.comparisons.html)看不同类型相互比较的例子。  **表格 10-4. 比较运算符**   | **例子** | **名称** | **结果** | | --- | --- | --- | | $a == $b | 等于 | **TRUE**，如果 $a 等于 $b。 | | $a === $b | 全等 | **TRUE**，如果 $a 等于 $b，并且它们的类型也相同。(PHP 4 only) | | $a != $b | 不等 | **TRUE**，如果 $a 不等于 $b。 | | $a <> $b | 不等 | **TRUE**，如果 $a 不等于 $b。 | | $a !== $b | 非全等 | **TRUE**，如果 $a 不等于 $b，或者它们的类型不同。(PHP 4 only) | | $a < $b | 小与 | **TRUE**，如果 $a 严格小于 $b。 | | $a > $b | 大于 | **TRUE**，如果 $a 严格 $b。 | | $a <= $b | 小于等于 | **TRUE**，如果 $a 小于或者等于 $b。 | | $a >= $b | 大于等于 | **TRUE**，如果 $a 大于或者等于 $b。 |   另外一个条件运算符是“?:”（或三元）运算符，它和 C 以及很多其它语言的操作一样。   |  | | --- | | <?php // Example usage for: Ternary Operator $action = (empty($\_POST['action'])) ? 'default' : $\_POST['action'];  // The above is identical to this if/else statement if (empty($\_POST['action'])) {     $action = 'default'; } else {     $action = $\_POST['action']; } ?> |   对于表达式 (expr1) ? (expr2) : (expr3)，如果 *expr1* 的值为 **TRUE**，则此表达式的值为 *expr2*，如果 *expr1* 的值为 **FALSE**，则此表达式的值为 *expr3*。  请参阅函数 [**strcasecmp()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.strcasecmp.html)、[**strcmp()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.strcmp.html) 及“[类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.html)”的有关章节。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.bitwise.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.errorcontrol.html) | | 位运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 错误控制运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.comparison.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.execution.html) | | |  | | | |
|  | **错误控制运算符**  PHP 支持一个错误控制运算符：@。当将其放置在一个 PHP 表达式之前，该表达式可能产生的任何错误信息都被忽略掉。  如果激活了 [track\_errors](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.errorfunc.html#ini.track-errors) 特性，表达式所产生的任何错误信息都被存放在变量 [$php\_errormsg](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/reserved.variables.html#reserved.variables.phperrormsg) 中。此变量在每次出错时都会被覆盖，所以如果想用它的话就要尽早检查。   |  | | --- | | <?php /\* Intentional file error \*/ $my\_file = @file ('non\_existent\_file') or     die ("Failed opening file: error was '$php\_errormsg'");  // this works for any expression, not just functions: $value = @$cache[$key]; // will not issue a notice if the index $key doesn't exist.  ?> |   **注:** @ 运算符只对[表达式](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.expressions.html)有效。对新手来说一个简单的规则就是：如果你能从某处得到值，你就能在它前面加上 @ 运算符。例如，你可以把它放在变量，函数和 [**include()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include.html) 调用，常量，等等之前。不能把它放在函数或类的定义之前，也不能用于条件结构例如 if 和 foreach 等。  参见 [**error\_reporting()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.error-reporting.html) 及手册中“[错误处理及日志函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.errorfunc.html)”的有关章节。  **注:** 错误控制前缀“@”不会屏蔽解析错误的信息。   |  | | --- | | **警告** | | 目前的“@”错误控制运算符前缀甚至使导致脚本终止的严重错误的错误报告也失效。这意味着如果你在某个不存在或类型错误的函数调用前用了“@”来抑制错误信息，那脚本会没有任何迹象显示原因而死在那里。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.comparison.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.execution.html) | | 比较运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 执行运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.errorcontrol.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.increment.html) | | |  | | | |
|  | **执行运算符**  PHP 支持一个执行运算符：反引号（``）。注意这不是单引号！PHP 将尝试将反引号中的内容作为外壳命令来执行，并将其输出信息返回（例如，可以赋给一个变量而不是简单地丢弃到标准输出）。使用反引号运算符“`”的效果与函数 [**shell\_exec()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.shell-exec.html) 相同。   |  | | --- | | <?php $output = `ls -al`; echo "<pre>$output</pre>"; ?> |   **注:** 反引号运算符在激活了 [安全模式](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.safe-mode.html#ini.safe-mode) 或者关闭了 [**shell\_exec()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.shell-exec.html) 时是无效的。  参见函数 [**popen()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.popen.html)、[**proc\_open()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.proc-open.html) 及手册中“[程序执行函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.exec.html)”和“[在命令行中使用 PHP](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/features.commandline.html)”的有关章节。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.errorcontrol.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.increment.html) | | 错误控制运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 加一／减一运算符 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.execution.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.logical.html) | | |  | | | |
|  | **加一／减一运算符**  PHP 支持 C 风格的前／后加一与减一运算符。  **表格 10-5. 加一／减一运算符**   | **例子** | **名称** | **效果** | | --- | --- | --- | | ++$a | 前加 | $a 的值加一，然后返回 $a。 | | $a++ | 后加 | 返回 $a，然后将 $a 的值加一。 | | --$a | 前减 | $a 的值减一， 然后返回 $a。 | | $a-- | 后减 | 返回 $a，然后将 $a 的值减一。 |   一个简单的示例脚本：   |  | | --- | | <?php echo "<h3>Postincrement</h3>"; $a = 5; echo "Should be 5: " . $a++ . "<br />\n"; echo "Should be 6: " . $a . "<br />\n";  echo "<h3>Preincrement</h3>"; $a = 5; echo "Should be 6: " . ++$a . "<br />\n"; echo "Should be 6: " . $a . "<br />\n";  echo "<h3>Postdecrement</h3>"; $a = 5; echo "Should be 5: " . $a-- . "<br />\n"; echo "Should be 4: " . $a . "<br />\n";  echo "<h3>Predecrement</h3>"; $a = 5; echo "Should be 4: " . --$a . "<br />\n"; echo "Should be 4: " . $a . "<br />\n"; ?> |   在处理字符变量的算数运算时，PHP 沿袭了 Perl 的习惯，而非 C 的。例如，在 Perl 中 'Z'+1 将得到 'AA'，而在 C 中，'Z'+1 将得到 '[' { ord('Z') == 90, ord('[') == 91 )。   |  |  | | --- | --- | | **例子 10-1. 涉及字符变量的算数运算**   |  | | --- | | <?php $i = 'W'; for($n=0; $n<6; $n++)   echo ++$i . "\n";  /\*   Produces the output similar to the following:  X Y Z AA AB AC  \*/ ?> | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.execution.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.logical.html) | | 执行运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 逻辑运算符 | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.increment.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html) | | |  | | | |
|  | **逻辑运算符**  **表格 10-6. 逻辑运算符**   | **例子** | **名称** | **结果** | | --- | --- | --- | | $a and $b | And（逻辑与） | **TRUE**，如果 $a 与 $b 都为 **TRUE**。 | | $a or $b | Or（逻辑或） | **TRUE**，如果 $a 或 $b 任一为 **TRUE**。 | | $a xor $b | Xor（逻辑异或） | **TRUE**，如果 $a 或 $b 任一为 **TRUE**，但不同时是。 | | ! $a | Not（逻辑非） | **TRUE**，如果 $a 不为 **TRUE**。 | | $a && $b | And（逻辑与） | **TRUE**，如果 $a 与 $b 都为 **TRUE**。 | | $a || $b | Or（逻辑或） | **TRUE**，如果 $a 或 $b 任一为 **TRUE**。 |   “与”和“或”有两种不同形式运算符的原因是它们操作的优先级不同。（见[运算符优先级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html#language.operators.precedence)。） | |  | |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.increment.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html) | | 加一／减一运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 字符串运算符 | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.logical.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.array.html) | | |  | | | | | |
|  | **字符串运算符**  有两个字符串运算符。第一个是连接运算符（“.”），它返回其左右参数连接后的字符串。第二个是连接赋值运算符（“.=”），它将右边参数附加到左边的参数后。更多信息见[赋值运算符](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.assignment.html)。   |  | | --- | | $a = "Hello "; $b = $a . "World!"; // now $b contains "Hello World!"  $a = "Hello "; $a .= "World!";     // now $a contains "Hello World!" |   请参阅手册中“[字符串类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.string.html)”和“[字符串函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.strings.html)”的有关章节。 | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.logical.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.array.html) | | 逻辑运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 数组运算符 | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html) | 章 10. 运算符 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html) | | |  | | | | | |
|  | **数组运算符**  PHP 仅有的一个数组运算符是 + 运算符。它把右边的数组附加到左边的数组后，但是重复的键值不会被覆盖。   |  | | --- | | $a = array("a" => "apple", "b" => "banana"); $b = array("a" =>"pear", "b" => "strawberry", "c" => "cherry");  $c = $a + $b;  var\_dump($c); |   执行后，此脚本会显示：   |  | | --- | | array(3) {   ["a"]=>   string(5) "apple"   ["b"]=>   string(6) "banana"   ["c"]=>   string(6) "cherry" } |   请参阅手册中“[数组类型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.types.array.html)”和“[数组函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.array.html)”的有关章节。 | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.string.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html) | | 字符串运算符 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.operators.html) | 流程控制 | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.elseif.html) | 章 11. 流程控制 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.while.html) | | |  | | | |
|  | **流程控制的替代语法**  PHP 提供了一些流程控制的替代语法，包括 if，while，for，foreach 和 switch。替代语法的基本形式是把左花括号（{）换成冒号（:），把右花括号（}）分别换成 endif;，endwhile;，endfor;，endforeach; 以及 endswitch;。   |  | | --- | | <?php if ($a == 5): ?> A is equal to 5 <?php endif; ?> |   在上面的例子中，HTML 内容“A is equal to 5”用替代语法嵌套在 if 语句中。该 HTML 的内容仅在 $a 等与 5 时显示。  替代语法同样可以用在 else 和 elseif 中。下面是一个包括 elseif 和 else 的 if 结构用替代语法格式写的例子：   |  | | --- | | <?php if ($a == 5):     print "a equals 5";     print "..."; elseif ($a == 6):     print "a equals 6";     print "!!!"; else:     print "a is neither 5 nor 6"; endif; ?> |   更多例子参见 [while](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.while.html)，[for](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.for.html) 和 [if](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html#control-structures.if)。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.elseif.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.while.html) | | elseif | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html) | while | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.declare.html) | 章 11. 流程控制 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.require.html) | | |  | | | |
|  | **return**  如果在一个函数中调用，[**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html) 语句立即结束此函数的执行并将它的参数作为函数的值返回。[**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html) 也会终止 [**eval()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.eval.html) 语句或者脚本文件的执行。  如果在全局范围中调用，则当前脚本文件中止运行。如果当前脚本文件是被 [**include()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include.html) 的或者 [**require()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.require.html) 的，则控制交回调用文件。此外，如果当前脚本是被 [**include()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include.html) 的，则 [**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html) 的值会被当作 [**include()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include.html) 调用的返回值。如果在主脚本文件中调用 [**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html)，则脚本中止运行。如果当前脚本文件是在 php.ini 中的配置选项 [auto\_prepend\_file](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.auto-prepend-file) 或者 [auto\_append\_file](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.directives.html#ini.auto-append-file) 所指定的，则此脚本文件中止运行。  更多信息见[返回值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html)。  **注:** 注意既然 [**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html) 是语言结构而不是函数，并*不*需要用括号将参数括起来。事实上不用括号比用括号更常见，尽管用哪一种并无所谓。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.declare.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.require.html) | | declare | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/control-structures.html) | [**require()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.require.html) | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include-once.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html) | | |  | | | |
|  | **章 12. 函数**  **目录**  [用户自定义函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html#functions.user-defined)  [函数的参数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html)  [返回值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html)  [变量函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.variable-functions.html)  [内部（内建）函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.internal.html)  **用户自定义函数**  一个函数可由以下的语法来定义：   |  |  | | --- | --- | | **例子 12-1. 展示函数用途的伪码**   |  | | --- | | <?php function foo ($arg\_1, $arg\_2, ..., $arg\_n) {     echo "Example function.\n";     return $retval; } ?> | |   任何有效的 PHP 代码都有可能出现在函数内部，甚至包括其它函数和 [类](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html#keyword.class) 定义。  在 PHP 3 中，函数必须在被调用之前定义。而 PHP 4 则不再有这样的 条件。*除非*函数如以下两个范例中有条件的定义。  如果一个函数以以下两个范例的方式有条件的定义，其定义必须在调用*之前*完成。   |  |  | | --- | --- | | **例子 12-2. 有条件的函数**   |  | | --- | | <?php  $makefoo = true;  /\* We can't call foo() from here    since it doesn't exist yet,    but we can call bar() \*/  bar();  if ($makefoo) {   function foo ()   {     echo "I don't exist until program execution reaches me.\n";   } }  /\* Now we can safely call foo()    since $makefoo evaluated to true \*/  if ($makefoo) foo();  function bar() {   echo "I exist immediately upon program start.\n"; }  ?> | |  |  |  | | --- | --- | | **例子 12-3. 函数中的函数**   |  | | --- | | <?php function foo() {   function bar()   {     echo "I don't exist until foo() is called.\n";   } }  /\* We can't call bar() yet    since it doesn't exist. \*/  foo();  /\* Now we can call bar(),    foo()'s processesing has    made it accessable. \*/  bar();  ?> | |   PHP 不支持函数重载，可能也不支持取消定义或者重定义已声明的函数。  **注:** 函数名是非大小写敏感的，不过在调用函数的时候，通常使用其在定义时相同的形式。  PHP 3 虽然支持默认参数（更多的信息请参照 [默认参数的值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html#functions.arguments.default)） ，但是却不支持可变的参数个数。 PHP 4 支持: 见 [可变长度的参数列表](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html#functions.variable-arg-list) 和涉及到的相关函数 [**func\_num\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-num-args.html)， [**func\_get\_arg()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-arg.html)， 和 [**func\_get\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-args.html) 以获取更多的信息。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include-once.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html) | | [**include\_once()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include-once.html) | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 函数的参数 | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | 章 12. 函数 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html) | | |  | | | |
|  | 函数的参数 通过参数列表可以传递信息到函数，该列表是以逗号作为分隔符的变量和常量列表。  PHP 支持按值传递参数（默认）, [通过引用传递](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html#functions.arguments.by-reference), 和 [默认参数值](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html#functions.arguments.default). 可变长度参数列表仅在 PHP 4 和后续版本中支持；更多信息请参照 [可变长度参数列表](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html#functions.variable-arg-list) 和涉及到的相关函数 [**func\_num\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-num-args.html)， [**func\_get\_arg()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-arg.html), 和 [**func\_get\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-args.html)。 PHP 3 中通过传递一个数组参数可以达到类似的效果：   |  |  | | --- | --- | | **例子 12-4. 向函数传递数组**   |  | | --- | | <?php function takes\_array($input) {     echo "$input[0] + $input[1] = ", $input[0]+$input[1]; } ?> | |  通过引用传递参数 缺省情况下，函数参数通过值传递（因而即使在函数内部改变参数的值，它 并不会改变函数外部的值）。如果你希望允许函数修改它的参数值，你必须 通过引用传递参数。  如果想要函数的一个参数总是通过引用传递，你可以在函数定义中该参数的 前面预先加上符号（&）：   |  |  | | --- | --- | | **例子 12-5. 用引用传递函数参数**   |  | | --- | | <?php function add\_some\_extra(&$string) {     $string .= 'and something extra.'; } $str = 'This is a string, '; add\_some\_extra($str); echo $str;    // outputs 'This is a string, and something extra.' ?> | |  默认参数的值 函数可以定义 C++ 风格的标量参数默认值，如下：   |  |  | | --- | --- | | **例子 12-6. 函数中默认参数的用途**   |  | | --- | | <?php function makecoffee ($type = "cappuccino") {     return "Making a cup of $type.\n"; } echo makecoffee (); echo makecoffee ("espresso"); ?> | |   上述片断的输出是：   |  | | --- | | Making a cup of cappuccino.  Making a cup of espresso. |   默认值必须是常量表达式，不是（比如）变量，类成员，或者函数调用。  请注意当使用默认参数时，任何默认参数必须放在任何非默认参数的右侧；否则， 可能函数将不会按照预期的情况工作。考虑下面的代码片断：   |  |  | | --- | --- | | **例子 12-7. 函数默认参数不正确的用法**   |  | | --- | | <?php function makeyogurt ($type = "acidophilus", $flavour) {     return "Making a bowl of $type $flavour.\n"; }  echo makeyogurt ("raspberry");   // won't work as expected ?> | |   上述例子的输出时：   |  | | --- | | Warning: Missing argument 2 in call to makeyogurt() in  /usr/local/etc/httpd/htdocs/php3test/functest.html on line 41  Making a bowl of raspberry . |   现在，比较上面的例子和这个例子：   |  |  | | --- | --- | | **例子 12-8. 函数默认参数正确的用法**   |  | | --- | | <?php function makeyogurt ($flavour, $type = "acidophilus") {     return "Making a bowl of $type $flavour.\n"; }  echo makeyogurt ("raspberry");   // works as expected ?> | |   这个例子的输出是：   |  | | --- | | Making a bowl of acidophilus raspberry. |  可变长度参数列表 PHP 4 已经在用户自定义函数中支持可变长度参数列表。这个真的很简单， 使用 [**func\_num\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-num-args.html)，[**func\_get\_arg()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-arg.html)，和 [**func\_get\_args()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.func-get-args.html) 函数.  无需特别的语法，参数列表仍然能够被明确无误的传递给函数并且正常运转。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html) | | 函数 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | 返回值 | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html) | 章 12. 函数 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.variable-functions.html) | | |  | | | |
|  | **返回值**  值通过使用可选的返回语句返回。任何类型都可以返回，其中包括列表和对象。 这导致函数立即结束它的运行，并且将控制权传递回它被调用的行。更多信息 请参照 [**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html)。   |  |  | | --- | --- | | **例子 12-9.** [**return()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.return.html) **函数的用法**   |  | | --- | | <?php function square ($num) {     return $num \* $num; } echo square (4);   // outputs '16'. ?> | |   函数不能返回多个值，但为了获得简单的结果，可以返回一个列表。   |  |  | | --- | --- | | **例子 12-10. 返回一个数组以得到多个返回值**   |  | | --- | | <?php function small\_numbers() {     return array (0, 1, 2); } list ($zero, $one, $two) = small\_numbers(); ?> | |   从函数返回一个引用，你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 & ：   |  |  | | --- | --- | | **例子 12-11. 由函数返回一个引用**   |  | | --- | | <?php function &returns\_reference() {     return $someref; }  $newref =& returns\_reference(); ?> | |   有关引用的更多信息, 请查看 [引用的解释](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html). |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.arguments.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.variable-functions.html) | | 函数的参数 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | 变量函数 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html) | 章 12. 函数 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.internal.html) | | |  | | | |
|  | **变量函数**  PHP 支持变量函数的概念。这意味着如果一个变量名后有圆括号，PHP 将寻找 与变量的值相同的函数，并且将尝试执行它。除了别的事情以外，这个可以被 用于实现回调函数，函数表等等。  变量函数不能用于语言结构，例如 [**echo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.echo.html)、[**print()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.print.html)、[**unset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unset.html)、[**isset()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.isset.html)、[**empty()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.empty.html)、[**include()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.include.html)、[**require()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.require.html) 以及类似的语句。您需要使用您自己的外壳函数来将这些结构用作变量函数。   |  |  | | --- | --- | | **例子 12-12. 变量函数示例**   |  | | --- | | <?php function foo() {     echo "In foo()<br>\n"; }  function bar($arg = '') {     echo "In bar(); argument was '$arg'.<br>\n"; }  // This is a wrapper function around echo function echoit($string) {     echo $string; }  $func = 'foo'; $func();        // This calls foo()  $func = 'bar'; $func('test');  // This calls bar()  $func = 'echoit'; $func('test');  // This calls echoit() ?> | |   您还可以利用变量函数的特性来调用一个对象的方法。   |  |  | | --- | --- | | **例子 12-13. 变量方法范例**   |  | | --- | | <?php class Foo {     function Var()     {         $name = 'Bar';         $this->$name(); // This calls the Bar() method     }      function Bar()     {         echo "This is Bar";     } }  $foo = new Foo(); $funcname = "Var"; $foo->$funcname();   // This calls $foo->Var()  ?> | |   请参阅 [**call\_user\_func()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.call-user-func.html)、[变量变量](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.variables.variable.html) 和 [**function\_exists()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.function-exists.html)。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.returning-values.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.internal.html) | | 返回值 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | 内部（内建）函数 | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.variable-functions.html) | 章 12. 函数 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | | |  | | | |
|  | **内部（内建）函数**  PHP 有很多标准的函数和结构。还有一些函数需要和特定地 PHP 扩展模块一起编译，否则在使用它们的时候就会得到一个致命的“未定义函数”错误。例如，要使用诸如 [**imagecreatetruecolor()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.imagecreatetruecolor.html) 的“[图像](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.image.html)函数”，您需要在编译 PHP 的时候加上 GD 的支持。或者，要使用 [**mysql\_connect()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.mysql-connect.html) 函数，您就需要在编译 PHP 的时候加上 [MySQL](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.mysql.html) 支持。另外还有一些核心函数，例如“[字符串函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.strings.html)”和“[变量函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.variables.html)”，它们存在于每一个版本的 PHP 中。调用 [**phpinfo()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.phpinfo.html) 或者 [**get\_loaded\_extensions()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.get-loaded-extensions.html) 可以得知 PHP 加载了那些扩展库。同时还应该注意，很多扩展库默认就是有效的。PHP 手册按照不同的扩展库组织了它们的文档。请参阅“[配置](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/configuration.html)”、“[安装](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/installation.html)”以及独立的扩展库章节以获取有关如何设置 PHP 的信息。  手册中“[如何阅读函数原型](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/missing-stuff.html#about.prototypes)”讲解了如何阅读和理解一个函数的原型。确认一个函数将返回什么，或者函数是否直接作用于传递的参数是很重要得。例如，[**str\_replace()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.str-replace.html) 函数将返回修改过的字符串，而 [**usort()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.usort.html) 却直接作用于传递的参数变量本身。手册中，每一个函数的页面中都有关于函数参数、行为改变、成功与否的返回值以及使用条件等信息。了解这些重要的（常常是细微的）差别是编写正确的 PHP 代码的关键。  请参阅函数 [**function\_exists()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.function-exists.html)、[函数的引用](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/funcref.html)、[**get\_extension\_funcs()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.get-extension-funcs.html) 和 [**dl()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.dl.html)。 | |  | |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.variable-functions.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | | 变量函数 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.html) | 类与对象 | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.internal.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.extends.html) | | |  | | | | | |
|  | **章 13. 类与对象**  **目录**  [类](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html#keyword.class)  [继承](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.extends.html)  [构造函数](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.constructor.html)  [::](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.paamayim-nekudotayim.html)  [parent](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.parent.html)  [序列化对象 － 会话中的对象](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.serialization.html)  [魔术函数 \_\_sleep 和 \_\_wakeup](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.magic-functions.html)  [构造函数中的引用](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.newref.html)  [PHP 4 中对象的比较](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php4.html)  [PHP 5 中对象的比较](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php5.html)  **类**  类是变量与作用于这些变量的函数的集合。使用下面的语法定义一个类：   |  | | --- | | <?php class Cart {     var $items;  // 购物车中的项目      // 把 $num 个 $artnr 放入车中      function add\_item ($artnr, $num)     {         $this->items[$artnr] += $num;     }      // 把 $num 个 $artnr 从车中取出      function remove\_item ($artnr, $num)     {         if ($this->items[$artnr] > $num) {             $this->items[$artnr] -= $num;             return true;         } else {             return false;         }     } } ?> |   上面的例子定义了一个 Cart 类，这个类由购物车中的商品构成的数组和两个用于从购物车中添加和删除商品的函数组成。   |  | | --- | | **警告** | | 你*不能*将一个类的定义放到多个文件中，或多个 PHP 块中。以下用法将不起作用：   |  | | --- | | <?php  class test {  ?>  <?php  function test() {  print 'OK';  }  }  ?> | |   以下警告仅用于 PHP 4。   |  | | --- | | **注意** | | 名称 stdClass 已经被 Zend 使用并保留。您不能在您的 PHP 代码中定义名为 stdClass 的类。 | | **注意** | | 函数名 \_\_sleep 和 \_\_wakeup 在 PHP 类中是魔术函数。除非想要与之联系的魔术功能，否则在任何类中都不能以此命名函数。 |  |  | | --- | | **注意** | | PHP 将所有以 \_\_ 开头的函数名保留为魔术函数。除非想要使用一些见于文档中的魔术功能，否则建议不要在 PHP 中将函数名以 \_\_ 开头。 |   在 PHP 4 中，var 变量的值只能初始化为常量。用非常量值初始化变量，您需要一个初始化函数，该函数在对象被创建时自动被调用。这样一个函数被称之为构造函数（见下面）。   |  | | --- | | <?php /\* PHP 4 中不能这样用 \*/ class Cart {     var $todays\_date = date("Y-m-d");     var $name = $firstname;     var $owner = 'Fred ' . 'Jones';     var $items = array("VCR", "TV"); }  /\* 应该这样进行 \*/ class Cart {     var $todays\_date;     var $name;     var $owner;     var $items;      function Cart()     {         $this->todays\_date = date("Y-m-d");         $this->name = $GLOBALS['firstname'];         /\* etc. . . \*/     } } ?> |   类也是一种类型，就是说，它们是实际变量的蓝图。必须用 new 运算符来创建相应类型的变量。   |  | | --- | | <?php $cart = new Cart; $cart->add\_item("10", 1);  $another\_cart = new Cart; $another\_cart->add\_item("0815", 3); ?> |   上述代码创建了两个 Cart 类的对象 $cart 和 $another\_cart，对象 $cart 的方法 add\_item() 被调用时，添加了 1 件 10 号商品。对于对象 $another\_cart，3 件 0815 号商品被添加到购物车中。  $cart 和 $another\_cart 都有方法 add\_item()，remove\_item() 和一个 items 变量。它们都是明显的函数和变量。你可以把它们当作文件系统中的某些类似目录的东西来考虑。在文件系统中，你可以拥有两个不同的 README.TXT 文件，只要不在相同的目录中。正如从为了根目录访问每个文件你需要输入该文件的完整的路径名一样，你必须指定需要调用的函数的完整名称：在 PHP 术语中，根目录将是全局名称空间，路径名符号将是 ->。因而，名称 $cart->items 和 $another\_cart->items 命名了两个不同的变量。注意变量名为 $cart->items，不是 $cart->$items，那是因为在 PHP 中一个变量名只有一个单独的美元符号。   |  | | --- | | <?php // 正确，只有一个 $ $cart->items = array("10" => 1);  // 不正确，因为 $cart->$items 变成了 $cart->"" $cart->$items = array("10" => 1);  // 正确，但可能不是想要的结果： // $cart->$myvar 变成了 $cart->items $myvar = 'items'; $cart->$myvar = array("10" => 1); ?> |   在一个类的定义内部，你无法得知使用何种名称的对象是可以访问的：在编写 Cart 类时，并不知道之后对象的名称将会命名为 $cart 或者 $another\_cart。因而你不能在类中使用 $cart->items。然而为了类定义的内部访问自身的函数和变量，可以使用伪变量 $this 来达到这个目的。$this 变量可以理解为“我自己的”或者“当前对象”。因而 '$this->>items[$artnr] += $num' 可以理解为“我自己的物品数组的 $artnr 计数器加 $num”或者“在当前对象的物品数组的 $artnr 计数器加 $num”。  **注:** 有一些不错的函数用来处理类和对象。你应该关注一下[类/对象函数库](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/ref.classobj.html)。 | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/functions.internal.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.extends.html) | | 内部（内建）函数 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 继承 | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.parent.html) | 章 13. 类与对象 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.magic-functions.html) | | |  | | | |
|  | **序列化对象 － 会话中的对象**  **注:** 在 PHP 3 中，在序列化和解序列化的过程中对象会失去类的关联。结果的变量是对象类型，但是没有类和方法，因此就没什么用了（就好像一个用滑稽的语法定义的数组一样）。   |  | | --- | | **注意** | | 以下信息仅在 PHP 4 中有效。 |   [**serialize()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.serialize.html) 返回一个字符串，包含着可以储存于 PHP 的任何值的字节流表示。[**unserialize()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unserialize.html) 可以用此字符串来重建原始的变量值。用序列化来保存对象可以保存对象中的所有变量。对象中的函数不会被保存，只有类的名称。  要能够 [**unserialize()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.unserialize.html) 一个对象，需要定义该对象的类。也就是，如果序列化了 page1.php 中类 A 的对象 $a，将得到一个指向类 A 的字符串并包含有所有 $a 中变量的值。如果要在 page2.php 中将其解序列化，重建类 A 的对象 $a，则 page2.php 中必须要出现类 A 的定义。这可以例如这样实现，将类 A 的定义放在一个包含文件中，并在 page1.php 和 page2.php 都包含此文件。   |  | | --- | | <?php // classa.inc:   class A   {       var $one = 1;        function show\_one()       {           echo $this->one;       }   }  // page1.php:   include("classa.inc");    $a = new A;   $s = serialize($a);   // 将 $s 存放在某处使 page2.php 能够找到   $fp = fopen("store", "w");   fputs($fp, $s);   fclose($fp);  // page2.php:   // 为了正常解序列化需要这一行   include("classa.inc");    $s = implode("", @file("store"));   $a = unserialize($s);    // 现在可以用 $a 对象的 show\_one() 函数了   $a->show\_one(); ?> |   如果在用会话并使用了 [**session\_register()**](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/function.session-register.html) 来注册对象，这些对象会在每个 PHP 页面结束时被自动序列化，并在接下来的每个页面中自动解序列化。基本上是说这些对象一旦成为会话的一部分，就能在任何页面中出现。  强烈建议在所有的页面中都包括这些注册的对象的类的定义，即使并不是在所有的页面中都用到了这些类。如果没有这样做，一个对象被解序列化了但却没有其类的定义，它将失去与之关联的类并成为 stdClass 的一个对象而完全没有任何可用的函数，这样就很没有用处。  因此如果在以上的例子中 $a 通过运行 session\_register("a") 成为了会话的一部分，应该在所有的页面中包含 classa.inc 文件，而不只是 page1.php 和 page2.php。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/keyword.parent.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.magic-functions.html) | | parent | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | 魔术函数 \_\_sleep 和 \_\_wakeup | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.magic-functions.html) | 章 13. 类与对象 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php4.html) | | |  | | | |
|  | **构造函数中的引用**  在构造函数中创建引用可能会导致混淆的结果。本节以教程形式帮助避免问题。   |  | | --- | | <?php class Foo {     function Foo($name)     {         // 在全局数组 $globalref 中建立一个引用         global $globalref;         $globalref[] = &$this;         // 将名字设定为传递的值         $this->setName($name);         // 并输出之         $this->echoName();     }      function echoName()     {         echo "<br>",$this->name;     }      function setName($name)     {         $this->name = $name;     } } ?> |   下面来检查一下用拷贝运算符 = 创建的 $bar1 和用引用运算符 =& 创建的 $bar2 有没有区别...   |  | | --- | | <?php $bar1 = new Foo('set in constructor'); $bar1->echoName(); $globalref[0]->echoName();  /\* 输出： set in constructor set in constructor set in constructor \*/  $bar2 =& new Foo('set in constructor'); $bar2->echoName(); $globalref[1]->echoName();  /\* 输出： set in constructor set in constructor set in constructor \*/ ?> |   显然没有区别，但实际上有一个非常重要的区别：$bar1 和 $globalref[0] 并没有被引用，它们不是同一个变量。这是因为“new”默认并不返回引用，而返回一个拷贝。  **注:** 在返回拷贝而不是引用中并没有性能上的损失（因为 PHP 4 及以上版本使用了引用计数）。相反更多情况下工作于拷贝而不是引用上更好，因为建立引用需要一些时间而建立拷贝实际上不花时间（除非它们都不是大的数组或对象，而其中之一跟着另一个变，那使用引用来同时修改它们会更聪明一些）。  要证明以上写的，看看下面的代码。   |  | | --- | | <?php // 现在改个名字，你预期什么结果？ // 你可能预期 $bar1 和 $globalref[0] 二者的名字都改了... $bar1->setName('set from outside');  // 但如同前面说的，并不是这样。 $bar1->echoName(); $globalref[0]->echoName();  /\* 输出为： set from outside set in constructor \*/  // 现在看看 $bar2 和 $globalref[1] 有没有区别 $bar2->setName('set from outside');  // 幸运的是它们不但相同，根本就是同一个变量。 // 因此 $bar2->name 和 $globalref[1]->name 也是同一个变量。 $bar2->echoName(); $globalref[1]->echoName();  /\* 输出为： set from outside set from outside \*/ ?> |   最后给出另一个例子，试着理解它。   |  | | --- | | <?php class A {     function A($i)     {         $this->value = $i;         // 试着想明白为什么这里不需要引用         $this->b = new B($this);     }      function createRef()     {         $this->c = new B($this);     }      function echoValue()     {         echo "<br>","class ",get\_class($this),': ',$this->value;     } }   class B {     function B(&$a)     {         $this->a = &$a;     }      function echoValue()     {         echo "<br>","class ",get\_class($this),': ',$this->a->value;     } }  // 试着理解为什么这里一个简单的拷贝会在下面用 \* // 标出来的行中产生预期之外的结果 $a =& new A(10); $a->createRef();  $a->echoValue(); $a->b->echoValue(); $a->c->echoValue();  $a->value = 11;  $a->echoValue(); $a->b->echoValue(); // \* $a->c->echoValue();  /\* 输出为： class A: 10 class B: 10 class B: 10 class A: 11 class B: 11 class B: 11 \*/ ?> | | |  | |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.magic-functions.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php4.html) | | 魔术函数 \_\_sleep 和 \_\_wakeup | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | PHP 4 中对象的比较 | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.newref.html) | 章 13. 类与对象 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php5.html) | | |  | | | | | |
|  | **PHP 4 中对象的比较**  在 PHP 4 中，对象比较的规则十分简单：如果两个对象的类相同，且它们有相同的属性和值，则这两个对象相等。类似的规则还适用与用全等符（===）对两个对象的比较。  如果我们执行以下范例中的代码： If we were to execute the code in the example below:   |  |  | | --- | --- | | **例子 13-1. PHP 4 中对象比较范例**   |  | | --- | | <?php function bool2str($bool) {     if ($bool === false) {             return 'FALSE';     } else {             return 'TRUE';     } }  function compareObjects(&$o1, &$o2) {     echo 'o1 == o2 : '.bool2str($o1 == $o2)."\n";     echo 'o1 != o2 : '.bool2str($o1 != $o2)."\n";     echo 'o1 === o2 : '.bool2str($o1 === $o2)."\n";     echo 'o1 !== o2 : '.bool2str($o1 !== $o2)."\n"; }  class Flag {     var $flag;      function Flag($flag=true) {             $this->flag = $flag;     } }  class SwitchableFlag extends Flag {      function turnOn() {         $this->flag = true;     }      function turnOff() {         $this->flag = false;     } }  $o = new Flag(); $p = new Flag(false); $q = new Flag();  $r = new SwitchableFlag();  echo "Compare instances created with the same parameters\n"; compareObjects($o, $q);  echo "\nCompare instances created with different parameters\n"; compareObjects($o, $p);  echo "\nCompare an instance of a parent class with one from a subclass\n"; compareObjects($o, $r); ?> | |   我们将得到：   |  | | --- | | Compare instances created with the same parameters  o1 == o2 : TRUE  o1 != o2 : FALSE  o1 === o2 : TRUE  o1 !== o2 : FALSE  Compare instances created with different parameters  o1 == o2 : FALSE  o1 != o2 : TRUE  o1 === o2 : FALSE  o1 !== o2 : TRUE  Compare an instance of a parent class with one from a subclass  o1 == o2 : FALSE  o1 != o2 : TRUE  o1 === o2 : FALSE  o1 !== o2 : TRUE |   这和我们按照比较规则推测的结果一致。当且仅当出自同一个类且属性及其值都相同的对象被认为是相等且相同的。  即时在有对象组合的时候，比较的规则也相同。在以下的范例中我们建立一个容器类来储存 **Flag** 对象的一个相关数组。   |  |  | | --- | --- | | **例子 13-2. PHP 4 中复合对象的比较**   |  | | --- | | <?php class FlagSet {     var $set;      function FlagSet($flagArr = array()) {         $this->set = $flagArr;     }      function addFlag($name, $flag) {         $this->set[$name] = $flag;     }      function removeFlag($name) {         if (array\_key\_exists($name, $this->set)) {             unset($this->set[$name]);         }     } }   $u = new FlagSet(); $u->addFlag('flag1', $o); $u->addFlag('flag2', $p); $v = new FlagSet(array('flag1'=>$q, 'flag2'=>$p)); $w = new FlagSet(array('flag1'=>$q));  echo "\nComposite objects u(o,p) and v(q,p)\n"; compareObjects($u, $v);  echo "\nu(o,p) and w(q)\n"; compareObjects($u, $w); ?> | |   我们得到预期的结果：   |  | | --- | | Composite objects u(o,p) and v(q,p)  o1 == o2 : TRUE  o1 != o2 : FALSE  o1 === o2 : TRUE  o1 !== o2 : FALSE  u(o,p) and w(q)  o1 == o2 : FALSE  o1 != o2 : TRUE  o1 === o2 : FALSE  o1 !== o2 : TRUE | | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.newref.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php5.html) | | 构造函数中的引用 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | PHP 5 中对象的比较 | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php4.html) | 章 13. 类与对象 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | | |  | | | |
|  | **PHP 5 中对象的比较**   |  | | --- | | **警告** | | 本扩展模块是*实验性*的。该模块的行为，包括其函数的名称以及其它任何关于此模块的文档可能会在没有通知的情况下随 PHP 以后的发布而改变。我们提醒您在使用本扩展模块的同时自担风险。 |   在 PHP 5 中，对象的比较比在 PHP 4 中更复杂，也比我们就某一面向对象语言期望的要多 （尽管 PHP 5 不是一种面向对象语言）。  当使用比较操作符（==）时，对象以一种很简单的规则比较：当两个对象有相同的属性和值，属于同一个类且被定义在相同的命名空间中，则两个对象相等。  另一方面，当使用全等符（===）时，当且仅当两个对象指向相同类（在某一特定的命名空间中）的同一个对象时才相等。  以下范例将阐明该规则。   |  |  | | --- | --- | | **例子 13-3. PHP 5 中对象比较范例**   |  | | --- | | <?php function bool2str($bool) {     if ($bool === false) {             return 'FALSE';     } else {             return 'TRUE';     } }  function compareObjects(&$o1, &$o2) {     echo 'o1 == o2 : '.bool2str($o1 == $o2)."\n";     echo 'o1 != o2 : '.bool2str($o1 != $o2)."\n";     echo 'o1 === o2 : '.bool2str($o1 === $o2)."\n";     echo 'o1 !== o2 : '.bool2str($o1 !== $o2)."\n"; }  class Flag {     var $flag;      function Flag($flag=true) {             $this->flag = $flag;     } }  namespace Other {      class Flag {         var $flag;          function Flag($flag=true) {                 $this->flag = $flag;         }     }  }  $o = new Flag(); $p = new Flag(); $q = $o; $r = new Other::Flag();  echo "Two instances of the same class\n"; compareObjects($o, $p);  echo "\nTwo references to the same instance\n"; compareObjects($o, $q);  echo "\nInstances of similarly named classes in different namespaces\n"; compareObjects($o, $r); ?> | |   该范例将输出：   |  | | --- | | Two instances of the same class  o1 == o2 : TRUE  o1 != o2 : FALSE  o1 === o2 : FALSE  o1 !== o2 : TRUE  Two references to the same instance  o1 == o2 : TRUE  o1 != o2 : FALSE  o1 === o2 : TRUE  o1 !== o2 : FALSE  Instances of similarly named classes in different namespaces  o1 == o2 : FALSE  o1 != o2 : TRUE  o1 === o2 : FALSE  o1 !== o2 : TRUE | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php4.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | | PHP 4 中对象的比较 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.html) | 引用的解释 | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php5.html) |  | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.whatdo.html) | | |  | | | |
|  | **章 14. 引用的解释**  **目录**  [引用是什么](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html#language.references.whatare)  [引用做什么](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.whatdo.html)  [引用不是什么](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.arent.html)  [引用传递](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html)  [引用返回](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html)  [取消引用](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.unset.html)  [引用定位](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.spot.html)  **引用是什么**  在 PHP 中引用意味着用不同的名字访问同一个变量内容。这并不像 C 的指针，它们是符号表别名。注意在 PHP 中，变量名和变量内容是不一样的，因此同样的内容可以有不同的名字。最接近的比喻是 Unix 的文件名和文件本身 － 变量名是目录条目，而变量内容则是文件本身。引用可以被看作是 Unix 文件系统中的紧密连接。 | |  | |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.oop.object-comparison-php5.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.whatdo.html) | | PHP 5 中对象的比较 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/langref.html) | 引用做什么 | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.arent.html) | | |  | | | | | |
|  | **引用做什么**  PHP 的引用允许你用两个变量来指向同一个内容。意思是，当你这样做时：   |  | | --- | | <?php $a =& $b ?> |   这意味着 $a 和 $b 指向了同一个变量。  **注:** $a 和 $b 在这里是完全相同的，这并不是 $a 指向了 $b 或者相反，而是 $a 和 $b 指向了同一个地方。  同样的语法可以用在函数中，它返回引用，以及用在 new 运算符中（PHP 4.0.4 以及以后版本）：   |  | | --- | | <?php $bar =& new fooclass(); $foo =& find\_var ($bar); ?> |   **注:** 不用 & 运算符导致对象生成了一个拷贝。如果你在类中用 $this，它将作用于该类当前的实例。没有用 & 的赋值将拷贝这个实例（例如对象）并且 $this 将作用于这个拷贝上，这并不总是想要的结果。由于性能和内存消耗的问题，通常你只想工作在一个实例上面。  尽管你可以用 @ 运算符来*关闭*构造函数中的任何错误信息，例如用 @new，但用 &new 语句时这不起效果。这是 Zend 引擎的一个限制并且会导致一个解析错误。  引用做的第二件事是用引用传递变量。这是通过在函数内建立一个本地变量并且该变量在呼叫范围内引用了同一个内容来实现的。例如：   |  | | --- | | <?php function foo (&$var) {     $var++; }  $a=5; foo ($a); ?> |   将使 $a 变成 6。这是因为在 foo 函数中变量 $var 指向了和 $a 指向的同一个内容。更多详细解释见[引用传递](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html)。  引用做的第三件事是[引用返回](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html)。 | | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.arent.html) | | 引用的解释 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 引用不是什么 | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.whatdo.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html) | | |  | | | |
|  | **引用不是什么**  如前所述，引用不是指针。这意味着下面的结构不会产生你预期的效果：   |  | | --- | | <?php function foo (&$var) {     $var =& $GLOBALS["baz"]; } foo($bar); ?> |   这将使 foo 函数中的 $var 变量在函数调用时和 $bar 绑定在一起，但接着又被重新绑定到了 $GLOBALS["baz"] 上面。不可能通过引用机制将 $bar 在函数调用范围内绑定到别的变量上面，因为在函数 foo 中并没有变量 $bar（它被表示为 $var，但是 $var 只有变量内容而没有调用符号表中的名字到值的绑定）。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.whatdo.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html) | | 引用做什么 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 引用传递 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.arent.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html) | | |  | | | |
|  | **引用传递**  你可以将一个变量通过引用传递给函数，这样该函数就可以修改其参数的值。语法如下：   |  | | --- | | <?php function foo (&$var) {     $var++; }  $a=5; foo ($a); // $a is 6 here ?> |   注意在函数调用时没有引用符号 － 只有函数定义中有。光是函数定义就足够使参数通过引用来正确传递了。  以下内容可以通过引用传递：   * 变量，例如 foo($a) * New 语句，例如 foo(new foobar()) * 从函数中返回的引用，例如：  |  | | --- | | <?php function &bar() {     $a = 5;     return $a; } foo(bar()); ?> |  * 详细解释见[引用返回](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html)。   任何其它表达式都不能通过引用传递，结果未定义。例如下面引用传递的例子是无效的：   |  | | --- | | <?php function bar() // Note the missing & {     $a = 5;     return $a; } foo(bar());  foo($a = 5) // 表达式，不是变量 foo(5) // 常量，不是变量 ?> |   这些条件是 PHP 4.0.4 以及以后版本有的。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.arent.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html) | | 引用不是什么 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 引用返回 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.unset.html) | | |  | | | |
|  | **引用返回**  引用返回用在当你想用函数找到引用应该被绑定在哪一个变量上面时。当返回引用时，使用此语法：   |  | | --- | | <?php function &find\_var ($param) {     /\* ...code... \*/     return $found\_var; }  $foo =& find\_var ($bar); $foo->x = 2; ?> |   本例中 find\_var 函数所返回的对象的属性将被设定（译者：指的是 $foo->x = 2; 语句），而不是拷贝，就和没有用引用语法一样。  **注:** 和参数传递不同，这里必须在两个地方都用 & 符号 － 来指出返回的是一个引用，而不是通常的一个拷贝，同样也指出 $foo 是作为引用的绑定，而不是通常的赋值。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.pass.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.unset.html) | | 引用传递 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 取消引用 | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.spot.html) | | |  | | | |
|  | **取消引用**  当你 unset 一个引用，只是断开了变量名和变量内容之间的绑定。这并不意味着变量内容被销毁了。例如：   |  | | --- | | <?php $a = 1; $b =& $a; unset ($a); ?> |   不会 unset $b，只是 $a。  再拿这个和 Unix 的 **unlink** 调用来类比一下可能有助于理解。 |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.return.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.spot.html) | | 引用返回 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 引用定位 | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **PHP 手册** | | | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.unset.html) | 章 14. 引用的解释 | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/security.html) | | |  | | | |
|  | 引用定位 许多 PHP 的语法结构是通过引用机制实现的，所以上述有关引用绑定的一切也都适用于这些结构。一些结构，例如引用传递和返回，已经在上面提到了。其它使用引用的结构有： global 引用 当用 **global $var** 声明一个变量时实际上建立了一个到全局变量的引用。也就是说和这样做是相同的：   |  | | --- | | <?php $var =& $GLOBALS["var"]; ?> |   这意味着，例如，unset $var 不会 unset 全局变量。 $this 在一个对象的方法中，$this 永远是调用它的对象的引用。 | |  |
| |  | | --- | |  | | |  |  |  | | --- | --- | --- | | [后退](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.unset.html) | [起点](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/index.html) | [前进](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/security.html) | | 取消引用 | [上一级](mk:@MSITStore:D:\PHP资料\黑客基地\php_manual_zh.chm::/zh/language.references.html) | 安全 | | | | | |