

目录

PHP 基础	2
PHP 代码标记	2
PHP 注释	2
PHP 语句分隔符	2
PHP 变量	3
常量	3
数据类型	4
流程控制	6
文件	7
函数	9
闭包	11
常用系统函数	12
错误处理	13
错误显示设置	15
字符串类型	17
字符串相关函数	19
数组	21
遍历数组	22
数组的相关函数	25

PHP 基础

PHP 是一种运行在服务器端的脚本语言，可以嵌入到 HTML 中。

PHP 代码标记

在 PHP 历史中，可以使用多种标记来区分 PHP 脚本。

ASP 标记: `<% php 代码 %>`

短标记: `<? php 代码 ? >` 简短风格 SGML (标准通用标记语言)

以上两种基本弃用，如果想使用需要在配置文件中开启。(配置文件: `php` 文件夹中的 `php.ini` 文件，搜索 `short_open_tag`, `asp_tags`)

脚本标记: `<script language="php">php 代码</script>`

标准标记 (常用): `<? php php 代码 ? >`

XML 风格 (可扩展标记语言)

PHP 注释

行注释: 一次注释一行

`//`: 后面内容为注释

块注释: 一次注释多行

`/*`:

中间内容全部都是注释

`*/`

PHP 语句分隔符

PHP 中，代码是以行为单位，一般以分号为行结束符。

特殊说明:

1. PHP 中标记结束符“`?>`”有自带语句结束符的效果，最后一行 PHP 可以没有语句结束符“`;`”。
2. PHP 中其实很多代码的书写并不是嵌入到 HTML 中，而是单独存在，通常书写习惯不建议使用标记结束符“`?>`”，PHP 会自动从开始到最后全部认为是 PHP 代码，从而解析。

PHP 变量

php 是一种动态网站开发的脚本语言，动态语言特点是交互性，会有数据的传递，而 PHP 作为“中间人”，需要进行数据的传递，传递的前提就是 PHP 能自己存储数据（临时存储）。

定义变量：在 PHP 中不需要任何关键字定义变量(赋值)；

PHP 中所有变量都必须使用“\$”符号。

PHP 中允许使用中文变量(不建议)。

删除变量：使用 unset(变量名)。

预定义变量

预定义变量：提前定义的变量，系统定义的变量，存储许多需要用到的数据（预定义变量都是数组）。

\$_GET: 获取所有表单以 get 方式提交的数据

\$_POST: 以 post 方式提交的数据

\$_REQUEST: GET 和 POST 提交的都保存

\$GLOBALS: PHP 中所有的全局变量

\$_SERVER: 服务器信息

\$_SESSION: session 会话数据

\$_COOKIE: cookie 会话信息

\$_ENV: 环境信息

\$_FILES: 用户上传的文件信息

可变变量：

```
$a = 'b';
```

```
$b = 'bb';
```

```
echo $a;//b
```

```
echo $$a;//bb—找到$a，解析结果：b，将前面的$符号与结果b绑定，：$b,然后解析。
```

变量传值：

值传递：\$新变量 = \$旧变量

引用传递：\$新变量 = &\$旧变量

常量

特点：数据不允许修改

常量只可以保存布尔值、整型、浮点型或字符串数据。

定义：

1. 使用定义常量的函数：define(‘常量名’,常量值);
2. PHP5.3 之后才有：const 常量名 = 值;

命名规则：

1. 常量不需要使用“\$”符号，一旦使用系统就会认为是变量。
2. 常量的名字组成由字母、数字和下划线组成，不能以数字开头。
3. 常量的名字通常是以大写字母为主。
4. 常量的命名规则比变量要松散，可以使用一些特殊字符，但只能使用 `define` 定义。

使用（访问）：

1. 普通常量：直接使用。
2. 特殊常量：`constant(‘常量名’)`;

系统常量：系统帮助用户定义的常量，用户可以直接使用。

运行 `phpinfo()`函数，可看到 PHP 预定义的许多常量。

常用的系统常量：

`PHP_VERSION`: PHP 版本号。

`PHP_INT_SIZE`:整型大小。

`PHP_INT_MAX`:整型能表示的最大值（PHP 中整形允许出现负数：带符号）。

系统魔术常量：双下划线开始+常量名+双下划线结束。魔术常量的值通常会跟着环境变化，但是用户改变不了。

`__DIR__`:当前被执行的脚本所在电脑的绝对路径

`__FILE__`:当前被执行的脚本所在电脑的绝对路径（带自己文件的名字）

`__LINE__`:当前所属的行数

`__NAMESPACE__`:当前所属的命名空间

`__CLASS__`:当前所属的类

`__METHOD__`:当前所属的方法

数据类型

数据类型（data type）指的是存储的数据本身的类型，而不是变量的类型，PHP 是一种弱类型语言，变量本身没有数据类型。

PHP 八大数据类型：

PHP 中将数据分为三大类，八小类。

简单(基本)数据类型	整型（int/integer）	占 4 个字节
	浮点型（float/double）	占 8 个字节
	字符串（string）	根据实际长度分配
	布尔类型（bool/boolean）	只有 true 和 false 两种取值
复合数据类型	对象类型（object）	存放对象（面向对象）
	数组类型（array）	存储多个数据（一次）

特殊数据类型	资源类型（resource）	存放资源数据（PHP 外部数据，如数据库、文件）
	空类型（NULL）	值就是一个 NULL,不能运算，对大小写不敏感。当被赋值为 NULL，或者尚未被赋值，或者被 unset()，变量都被认为 NULL
	Callable 类型	可以传递给其他函数的函数

类型转换：

当前数据类型→目标数据类型

1. 自动转换：系统根据需求自己判定。
2. 强制（手动）转换：人为根据需要的目标类型转换。
强制转换规则：在变量之前加一个括号()，然后在里面写上目标数据类型。
NULL 类型用到 unset()

其他类型转布尔类型：true 或者 false。

其他类型转数值：

1. 布尔 true 为 1，false 为 0。
2. 字符串转数值：
 - a) 以字母开头的字符串，永远为 0。
 - b) 以数字开头的字符串，取到碰到字母为止，（不会同时包含两个小数点）。

类型判断：通过一组以 is_类型名的函数来判断：is_XXX(变量名),返回结果为布尔类型（true/false）。Bool 类型不能用 echo 来查看，可以使用 var_dump 结构查看。

var_dump(变量 1，变量 2，...)—会将数据类型和值同时打印。

empty():判断数据的值是否为“空”，不是 NULL,为空则返回 true。（判断变量的值）

isset():判断数据存储的变量本身是否存在，存在变量返回 true。（判断变量是否存在）

获取和设定数据（变量）的类型：

gettype(变量名): 获取类型，得到的是该类型对应的字符串。（不推荐使用来测试某种类型，推荐使用 is_*函数代替）

settype(变量名，数据类型): 设定数据类型，与强制转换不同。

1. 强制类型转换，是对数据复制的内容进行处理，（不会处理实际存储的内容），数据原来是什么类型，转换后仍是什么类型。
2. settype 会直接改变数据本身。

进制转换函数：

decbin(): 十进制转二进制

decoct(): 十进制转八进制

dechex(): 十进制转十六进制

bindec(): 二进制转十进制

octdec(): 八进制转十进制

bin2hex():二进制转十六进制

...

连接运算符：是 PHP 中将多个字符串拼接的一种符号。

.:将两个字符串连接到一起。

.=: 复合运算，将左边的内容与右边的内容连接起来，然后重新复制给左边变量。

错误抑制符：有一些错误提前可以预知，又无法避免，但又不希望给用户看，可以用错误抑制符。

@: 在可能出错的表达式前面使用@符号即可。

使用错误抑制符，如果已经在 php.ini 文件中启用了 PHP 的 track_errors 特性，错误信息将会保存在全局变量 \$php_errormsg 中。

执行操作符：反向单引号。位于与~同样的位置。

PHP 会试着将反向单引号之间的命令当做服务器命令行命令来执行。表达式的值就是命令的执行结果。

```
$out = `dir c:`;  
echo '<pre>'.$out.'</pre>';
```

得到一个目录列表，并将该列表保存在\$out中

流程控制

流程控制：代码执行的方向。

控制分类：

顺序结构：代码从上到下，顺序执行。（代码执行的最基本结构）

分支结构：给定一条件，同时有多种可执行代码（块），然后根据条件执行某一段代码。

PHP 中分支结构主要有两种：if 分支和 switch 分支。

循环结构：在某个条件控制范围内，指定的代码（块）可以重复执行。

For 循环，while 循环、do-while 循环，foreach 循环（专门针对数组）。

循环控制：

中断控制：重新开始循环。

continue 层级 //默认为 1

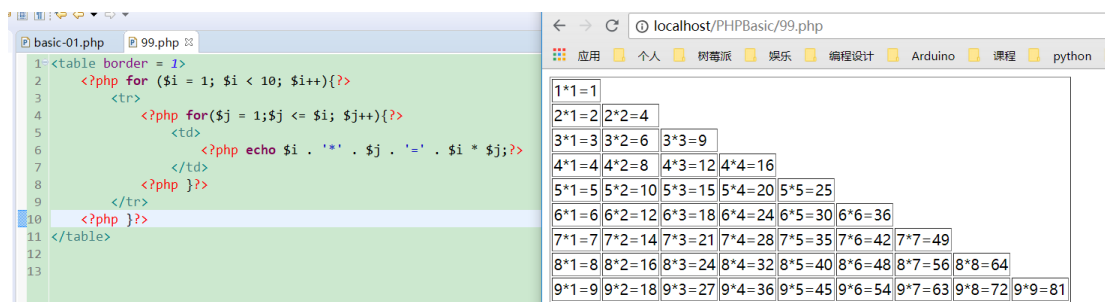
终止控制：循环直接进结束。

break 层级 // 默认为 1

流程控制替代语法：

流程控制替代语法：分支和循环结构的替代语法。

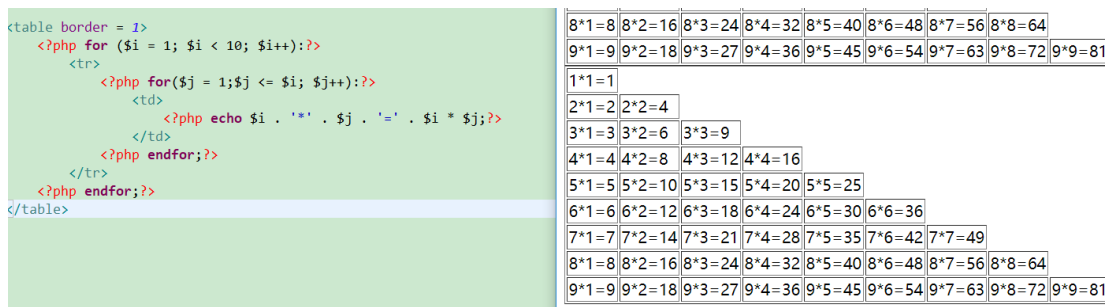
PHP 与 HTML 混搭。



在 PHP 中这些大括号不美观，所以 PHP 提供了一种替代机制。

for(;;){ → for(;;):

} → endfor;



PHP 的替代语法: PHP 应该在 HTML 中只做数据输出, 输出通常伴有条件判断和循环操作, 因此, PHP 提供了对应分支结构和循环结构的替代语法, 全部都是对应的一个模式:

左大括号 { 使用冒号 : 替代

右大括号 } 使用 end+对应的起始标记替代

文件

文件包含四种形式: **include, require, include_once, require_once**

- require 函数通常放在 PHP 程序的最前面, PHP 程序在执行前, 就会先读入 require 所指定引入的文件, 使它变成 PHP 程序网页的一部份。
- include 函数一般放在流程控制的处理部分中。PHP 程序网页在读到 include 的文件时, 才将它读进来。这种方式, 可以把程序执行时的流程简单化。
- **include 有返回值, 而 require 没有。**
- “require”的工作方式是为了让 PHP 程序得到更高的效率, 当它在同一个 PHP 网页中解释过一次后, 第二次出现便不会再解释, 这是它的优点。不过严格来说, 这也是它的唯一的缺点, 因为它不会重复解释引入的文件, 所以 **当 PHP 网页中使用循环或条件语句来引入文件时, “require”则不会做任何的改变。**当有类似这样的情形时, 就必须使用“include”命令来引入 文件了。

- 当 PHP 遇到一个利用“include”方式引入的文件，它就会解释一次；遇到第二次时，PHP 还是会重新解释一次。与“require”相比，“include”的执行效率则会下降许多；而且当引入文件中包含了用户自定义的函数时，PHP 在解释的过程中会发生函数重复定义的问题。不过“include”也不是没有优点的，因为在 PHP 网页中，它会每遇到一次“include”命令就会重复解释一次，所以非常适合使用在循环或条件判断的语句里。
- “include_once()”函数和“require_once()”函数，会先检查目标档案的内容是不是在之前就已经导入过了，如果是，便不会再次重复导入同样的内容。

文件加载原理：

1. 在文件加载（include 或者 require）的时候，系统会自动的将包含文件的代码相当于嵌入到当前文件中
 2. 加载位置：在哪加载，对应的文件中代码嵌入的位置就是对应的 include 位置
 3. 在 PHP 中被包含的文件是单独进行编译的
- PHP 文件在编译过程中如果出现了语法错误，那么会失败（不会执行）；但是如果被包含的文件有错误时，系统会执行到包含 include 这条语句的时候才会报错。

include 和 require 区别：

include 和 include_once 的区别：

include ：系统碰到一次，执行一次，如果对同一个文件进行多次加载，那么系统会执行多次；

include_once ：系统碰到多次，也只会执行一次。

require 和 include 的区别：本质都是包含文件，唯一的区别在于包含不到文件的时候，报错的形式不一样。

include 的错误级别比较轻，不会阻止代码执行。

require 要求较高，如果包含出错，require 后面的代码不会执行。

文件加载路径：

文件在加载的时候需要指定文件路径才能保证 PHP 正确的找到对应的文件。

文件加载路径包含两大类：

1. 绝对路径：
 - 从磁盘的根目录开始（本地绝对路径）
 - Windows：盘符 c:/路径/PHP 文件
 - Linux：/路径/PHP 文件
 - 从网站根目录开始（网络绝对路径）
 - /：相对于网站主机名字对应的路径
 - localhost/index.php → D:/Server/Apache24/htdocs/PHPBasic/ PHP 文件
2. 相对路径：从当前文件所在的目录开始的路径
 - .或者./:表示当前文件夹
 - ../：上级目录（当前文件夹的上一层文件夹）

绝对路径与相对路径的加载区别：

1. 绝对路径相对效率偏低，但是相对安全（路径不会出问题）。
2. 相对路径相对效率高些，但是容易出错（相对路径会发生改变）。

函数

PHP 中函数名不区分大小写。

函数：function

函数定义语法：

函数的几个对应的关键点：function，关键字、函数名、参数（形参和实参）、函数体和返回值。

基本语法：

```
function 函数名（参数）{  
    //函数体  
    //返回值 return 结果;  
}
```

定义函数的目的：是为了实现代码的重复利用。一个功能一个函数（简单明了）

函数调用特点：只要系统在内存中能够找到对应的函数，就可以执行（[函数的调用可以在函数定义前](#)）

函数返回值

返回值：return，指的是将函数实现的结果通过 return 关键字，返回给函数外部（函数调用处），在 PHP 中所有函数都有函数值（如果没有明确 return 使用，那么系统默认返回 NULL）。

return 作用

- 1、return 在函数内部的价值：返回当前函数的结果（当前函数运行结束）
- 2、return 还可以在文件中直接使用（不在函数中），代表文件将 return 结果转交给包含当前文件的位置。（通常在系统配置文件中使用的较多），在文件中也代表终止文件后面的代码，return 之后的内容不会执行。

PHP 作用域

- 1、全局变量：就是用户普通定义的变量（函数外部定义）。

用户定义的所有全局变量都会放在 \$GLOBALS 中，**局部作用域不能访问全局变量**，\$GLOBALS 是一个超全局变量，可以任意地方访问。

所属全局空间：在 PHP 中只允许在全局空间使用，理论上函数内部不可访问。

脚本周期：直到脚本运行结束（最后一行代码执行完）

- 2、局部变量：就是在函数内部定义的变量

所属当前函数空间：在 PHP 中只允许在当前函数自己内部使用

函数周期：函数执行结束（函数是在栈区中开辟独立内存空间运行）

3、超全局变量：系统定义的变量（预定义变量：\$_SERVER_、\$_POST_等）

所属超全局空间：没有访问限制（函数内外都可以访问）

超全局变量会将全局变量自动纳入到\$GLOBALS 里面，而\$GLOBALS 没有作用域限制，所以能够帮助局部去访问全局变量，但是必须使用数组方式。（不常用）

如果想在函数内部使用外部变量：除了\$GLOBALS 外，可以通过参数传值。

Global 关键字

global 关键字：实现全局访问局部，同时局部也可以访问全局

global 关键字是一种在函数里面定义变量的一种方式。

1、如果使用 global 定义的变量名在外部存在（全局变量），那么系统在函数内部定义的变量直接指向外部全局变量所指向的内存空间（同一个变量）

2、如果使用 global 关键字定义的变量名在外部不存在（全局变量），系统会自动在全局空间（外部）定义一个与局部变量同名的全局变量

本质形式：在函数的外部和内部，对一个同名变量（全局和局部）使用同一块内存地址保存数据，从而实现共同拥有。

基本语法：

```
global 变量名; //不能赋值
```

```
变量名 = 值; //修改
```

静态变量

静态变量 static，是在函数内部定义的变量（属于局部变量），使用 static 关键字来修饰，用来实现**跨函数共享数据（同一个函数被多次调用）**的变量：函数运行结束所有局部变量都会清空，如果重新运行函数，所有局部变量又会重新初始化。

基本语法：

```
function 函数名 () {  
    static 变量名 = 值; // 通常会在定义的时候就直接赋值  
}
```

静态变量的原理：系统在进行编译的时候就会对 static 这一行进行初始化：为静态变量赋值，函数在调用的时候，会自动跳过 static 关键字这一行。

静态变量的使用：

1、为了统计：当前函数被调用的次数

2、为了统筹函数多次调用得到的不同结果（递归思想）

可变函数

可变函数：当前有一个变量所保存的值，刚好是一个函数的名字，那么就可以使用变量+() 来充当函数名使用。

基本语法：

```
$变量 = 'display';
```

```
function display(){

}
```

```
//可变函数
$变量();
```

可变函数在系统使用的过程中还是比较多的，尤其是使用很多系统函数的时候，需要用户在外定义一个自定义函数，但是需要传入到系统函数内部使用。

```
$func = 'display';

//定义系统函数（假设）
function sys_function($arg1,$arg2){
    //给指定的函数（第一个参数），求对应的第二个参数的四次方
    //为实际用户输入的数值进行处理
    $arg2 = $arg2 + 10;
    return $arg1($arg2); //user_function(20);
}
//定义一个用户函数，求一个数的四次方
function user_function($num){
    return pow($num, 4);
}
echo sys_function('user_function', 10);
```

将一个用户定义的函数传入给另一个函数（函数名）去使用的过程，称之为回调过程，而被传入的函数称之为回调函数

匿名函数

匿名函数：没有名字的函数。

基本语法：

```
变量名 = function(){

};
变量名(); //调用
```

变量保存匿名函数，本质得到的是一个对象（closure）；

闭包

闭包：closure，一词来自于以下两者的结合：要执行的代码块（由于自变量被包含在代码块中，这些自由变量以及它们引用的对象没有被释放）和为自由变量提供绑定的计算环境（作用域）。

简单理解：函数内部有一些局部变量（要执行的代码块）在函数执行之后没有被释放，是因为在函数内部还有对应的函数在引用（函数的内部函数：匿名函数）。

```

<?php
function display(){
    $name = __FUNCTION__;
    //定义匿名函数
    $innerfunction = function() use($name){//use就是将外部变量（局部）保留给内部使用（闭包）
        echo $name;
    };
    //调用函数
    $innerfunction();
    //返回内部匿名函数
    // return $innerfunction;
}
// $closure = display();
// $closure();
display();

```

证明：函数的局部变量在函数使用之后没有被释放？

- 1、使用内部匿名函数；
- 2、匿名函数使用局部变量：use 关键字；
- 3、匿名函数被返回给外部使用；

伪类型

伪类型：假类型，实际上在 PHP 中不存在的类型，但是通过伪类型可以帮助程序员更好的查看操作手册从而方便学习。

伪类型主要有两种：在三大类八小类之外。

mixed：混合的，可以是多种 PHP 中的数据类型

number：数值的，可以是任意数值类型（整型和浮点型）

常用系统函数

1) 有关输出的函数

echo:输出内容（非函数），输出字符串，可以输出多个字符串。

print():类似于 echo 输出提供的内容，本质是一种结构（不是函数），返回 1，可以不需要使用括号。只能输出一个字符串。

print_r():类似于 var_dump，但是比 var_dump 简单，不会输出数据的类型，只会输出值（数组打印使用比较多）

var_dump：可以输出任何内容和其类型。

printf(): 格式化输出

2) 有关时间的函数

date():按照指定格式对对应的时间戳（从 1970 年格林威治时间开始计算的秒数），如果没有指定特定的时间戳，那么就是默认解释当前时间戳。

time():获取当前时间对应的的时间戳。

microtime():获取微妙级别的时间。

strtotime():按照规定格式的字符串转换成时间戳。

```
//时间函数
echo date('Y 年 m 月 d 日 H:i:s',55555555),'<br />';
echo time(),'<br />';
echo microtime(),'<br />';
echo strtotime('tomorrow 10 hours');
```

3) 有关数学的函数

max():指定参数中最大的值(多个数或者数组)。

min():比较两个数中较小的值。

rand():得到一个随机数, 指定区间的随机整数。

mt_rand(): 与 rand 一样, 只是底层结构不一样, 效率比 rand 高 (建议使用)。

round():四舍五入。

ceil():向上取整。

floor():向下取整。

pow():求指定数字的指定次结果。

abs():取绝对值

sqrt():求平方根

4) 有关函数的函数

function_exists():判断指定的函数名字是否在内存中存在, 帮助用户不去使用一个不存在的函数。

func_get_arg():在自定义函数中去获取指定数值对应的参数。

func_get_args():在自定义函数中去获取所有的参数, 得到的是一个数组

func_num_args(): 获取当前自定义函数的参数数量

```
18 echo '<hr /><pre>';
19
20 function test($a,$b){
21     //获取指定参数
22     var_dump(func_get_arg(1));
23     //获取所有参数
24     var_dump(func_get_args());
25     //获取参数数量
26     var_dump(func_num_args());
27 }
28 //调用函数
29 function_exists('test') && test(1,'2',3,4);//存在执行 (短路)
30
```

string(1) "2"
array(4) {
 [0]=> int(1)
 [1]=> string(1) "2"
 [2]=> int(3)
 [3]=> int(4)
}

参数的标识是从0 开始

func_get_args和 func_num_args都是统计对应实参的数量

错误处理

错误处理: 指的是系统 (或者用户) 在对某些代码进行执行的时候, 发现有错误, 就会通过错误处理的形式告知程序员。

错误分类

1) 语法错误: 用户书写的代码不符合 PHP 语法规范, 语法错误会导致代码在编译过程中不

通过，所以代码不会执行（parse error）

2）运行时错误：代码编译通过，但是代码在执行过程中会出现一些条件不满足导致的错误（runtime error）

3）逻辑错误：程序员在写代码的时候不够规范，出现了一些逻辑性的错误，导致代码正常执行，但是得不带想要的结果

错误代号

所有看到的错误代号在 PHP 中都被定义成立系统常量（可以直接使用）

1）系统错误：

E_PARSE:编译错误，代码不会执行

E_ERROR: fatal error, 致命错误，会导致代码不能正确继续执行（出错的位置断掉）

E_WARNING: warning, 警告错误，不会影响代码执行，但是可能得到意想不到的结果。

E_NOTICE: notice, 通知错误，不会影响代码执行

2）用户错误：E_USER_ERROR、E_USER_WARNING、E_USER_NOTICE

用户在使用自定义错误触发的时候，会使用到的错误代号（系统不会用到）

3）其他：E_ALL，代表着所有的错误，（通常在进行错误控制的时候使用比较多），建议在开发过程中（开发环境）使用

所有以 E 开头的错误常量（代号）其实都是由一个字节存储，然后每一种错误占据一个对应的位，如果想进行一些错误的控制，可以使用位运算进行操作。

如：

排除通知级别 notice: E_ALL & ~E_NOTICE

只要警告和通知: E_WARNING | E_NOTICE

错误触发

程序运行时触发：系统自动根据错误发生后，对比对应的错误信息，输出给用户，主要针对代码的语法错误和运行时错误。

人为触发：知道某些逻辑可能会出错，从而使用对应的判断代码来触发响应的错误提示。

trigger_error(‘提示信息’ [,错误级别])，可以通过第二个参数进行严格性控制。

```
// echo $a;
header('Content-type:text/html;charset=utf-8'); //指定编码格式
$b = 0;
if($b==0){
    //人为触发
    trigger_error('除数不能为0! '); //默认notice,会继续执行
    trigger_error('除数不能为0! ',E_USER_ERROR); //默认error,不会继续执行
}
echo 'hello';
```

错误显示设置

错误显示设置：哪些错误该显示，以及如何显示

在 PHP 中，其实有两种方式来设置当前脚本的错误处理。

1、PHP 的配置文件：全局配置，php.ini 文件

display_errors:是否显示错误

error_reporting:显示什么级别的错误

```
; http://php.net/error-reporting
error_reporting = E_ALL 默认
; http://php.net/display-errors
display_errors = On 默认
```

2、在运行的 PHP 脚本中去设置：在脚本中定义的配置项级别比配置文件高（通常在开发当中都会在代码中去进行控制和配置）

error_reporting;设置对应的错误显示级别

ini_set('配置文件中的配置项',配置值)

ini_set('error', E_ALL);

ini_set('display_errors',1);

错误日志设置

在实际生产环境中，不会直接让错误赤裸裸的展示给用户：

1、不友好

2、不安全，错误会暴露网站很多信息（路径、文件名）

但错误不可避免，又希望捕捉到让后台程序员去修改：需要保存到日志文件中，需要在 PHP 配置文件中或者代码中（ini_set）设置对应的 error_log 配置项。

1、开启日志功能

```
487 ; directive found below. While errors
488 ; servers they should still be monitored
489 ; Default Value: Off
490 ; Development Value: On
491 ; Production Value: On
492 ; http://php.net/log-errors
493 log_errors = On
494
```

2、指定路径

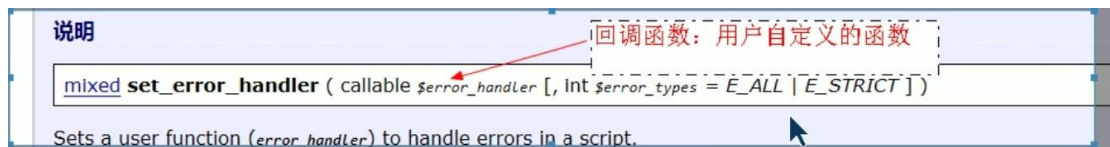
```
; http://php.net/error-log
; Example:
;error_log = php_errors.log
; Log errors to syslog (Event Log on Windows).
;error_log = syslog
error_log = 'D:/Server/php7/errlog/php_errors.log'
;windows.show_crt_warning
```


自定义错误处理

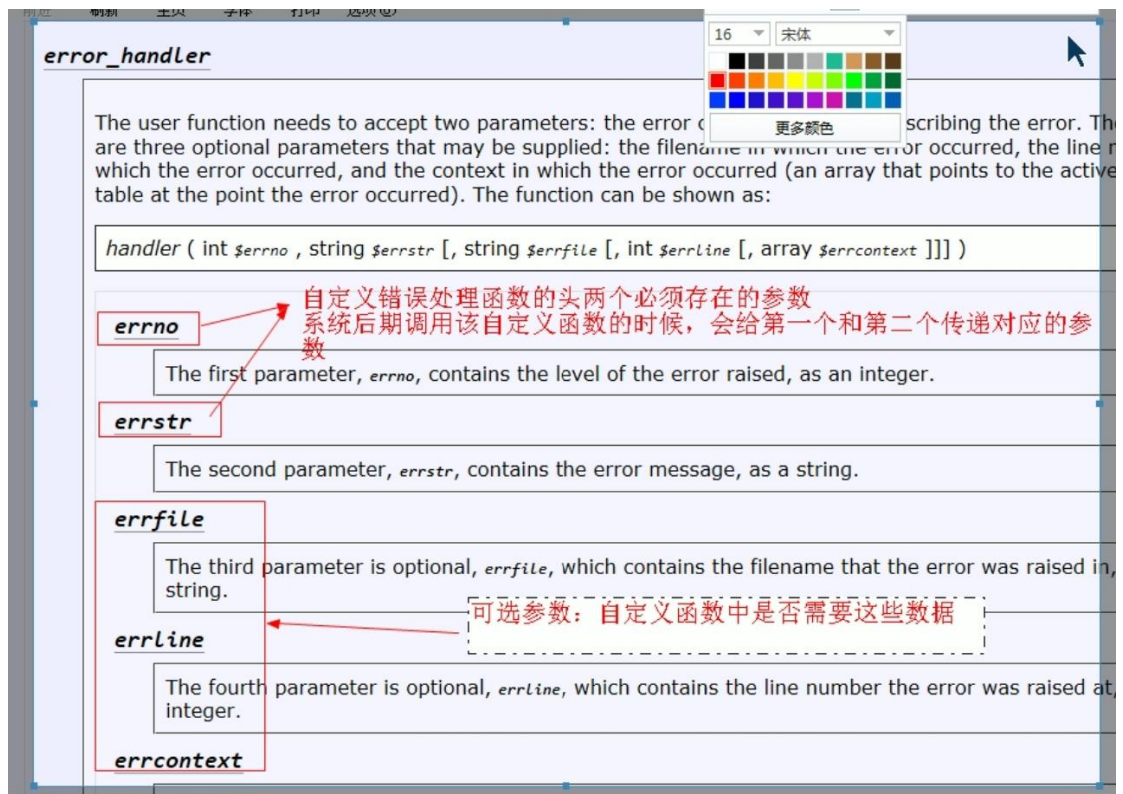
最简单的错误处理：`trigger_error()`函数，但是该函数不会阻止系统报错。

PHP 系统提供了一种用户处理错误的机制：用户自定义错误处理函数，然后将该函数增加到系统错误处理的句柄中，然后系统会在碰到错误之后，使用用户定义的错误函数。

1、如何将用户自定义的函数放到系统中？`set_error_handler()`



2、自定义错误处理函数，系统有要求。



代码实现：

1、自定义错误处理函数：注意参数


```

*/
//自定义函数
function my_error($errno,$errstr,$errfile,$errline){
    //判断：当前会碰到的错误有哪些
    if(!(error_reporting() & $errno)) { //error_reporting没有参数代表获取当前系统错误处理对应的级别
        return false; //系统中不存在则不能处理
    }

    //开始判断错误类型
    switch ($errno){
        case E_ERROR:
        case E_USER_ERROR:
            echo 'fatal error in file' . $errfile . 'on line' . $errline . '<br />';
            echo 'error info :' . $errstr;
            break;
        case E_WARNING:
        case E_USER_WARNING:
            echo 'Warning error in file' . $errfile . 'on line' . $errline . '<br />';
            echo 'error info :' . $errstr;
            break;
        case E_NOTICE:
        case E_USER_NOTICE:
            echo 'Notice error in file' . $errfile . 'on line' . $errline . '<br />';
            echo 'error info :' . $errstr;
            break;
    }
}

```

排除当前系统本身就要排除的错误

错误匹配

2、注册自定义函数：修改错误处理机制

```

34 //报错
35 echo $a;
36 //修改错误机制
37 set_error_handler('my_error');
38
39 echo $a;

```

Notice: Undefined variable: a in D:\Server\Apache24\htdocs\PHPBasic\error_mine.php on line 36
Notice error in fileD:\Server\Apache24\htdocs\PHPBasic\error_mine.php on line41
error info :Undefined variable: a

当前属于简单自定义模式，如果要复杂，可以在某些影响代码功能的错误发生后，让用户跳转到某个指定界面。

字符串类型

- 1) 单引号字符串：使用单引号包裹 单引号里面的不进行解释，直接输出。
- 2) 双引号字符串：使用双引号包裹 双引号里面的字段会经过编译器解释，然后再当作 HTML 代码输出。

引号方式：比较适合定义那些比较短（不超过一行）或者没有结构要求的字符串，如果有结构要求，或者内容超过一行，可以使用以下两种结构来定义：

- 3) nowdoc 字符串：没有单引号的单引号字符串

```
$str = <<<'边界符'
```

字符串内容

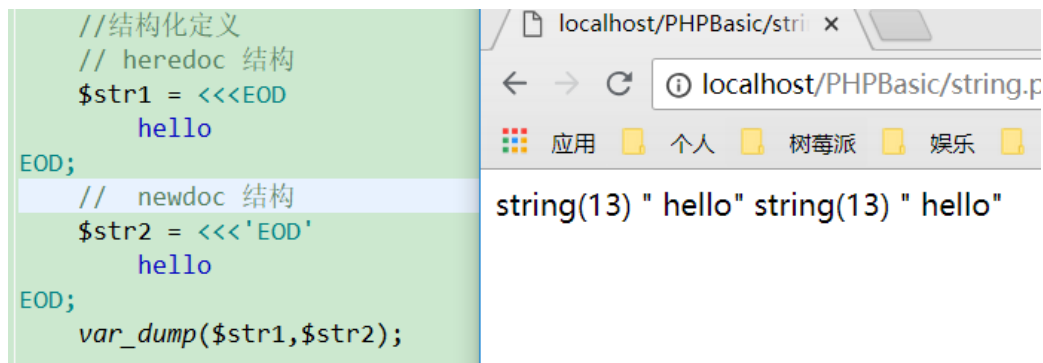
```
边界符;
```

- 4) heredoc 字符串：没有双引号的双引号字符串

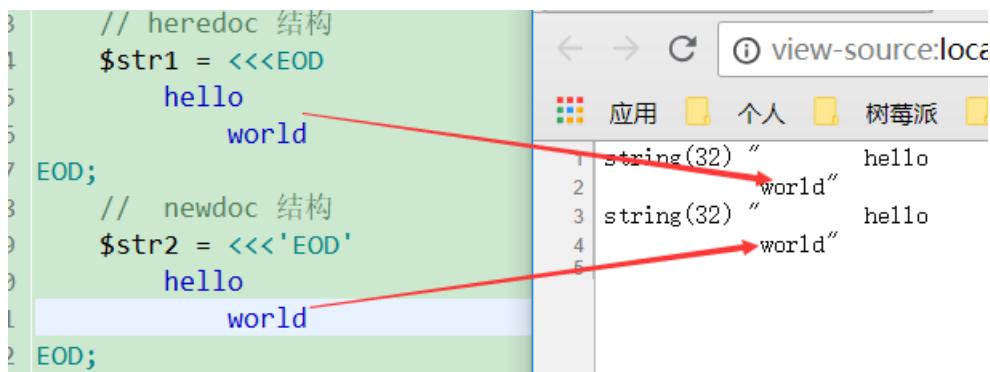
```
$str = <<<边界符
```

字符串内容

```
边界符;
```



heredoc 和 newdoc 比引号区别多一点。



结构化定义字符串变量的规则：（heredoc 和 newdoc）

1、结构化定义字符串对应的边界符有条件：

- 1.1 上边界符后面不能跟任何内容（空格、注释也不能跟）
- 1.2 下边界符必须顶格（最左边）
- 1.3 下边界符同样后面只能跟分号，不能跟其他任何内容

2、结构化定义字符串的内部（边界符之间）的所有内容都是字符串本身（包括注释）。会当字符串全部输出。

字符串转义

转义的含义：在计算机通用协议中，有一些特定的方式定义的字母，系统会特定处理，通常这种方式都是使用反斜杠+字母（单词）的特性。

\r\n:回车换行。

PHP 转义字符：

\' ：在单引号字符串中显示单引号

\" ：在双引号字符串中显示双引号

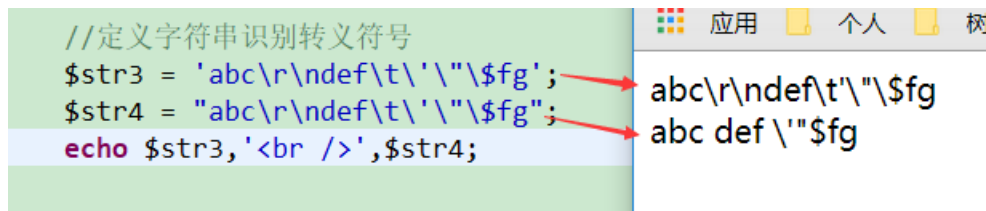
\\$ ：在 PHP 中使用\$符号作为变量符号，因此需要特别识别。

\r \n \t

单引号和双引号的区别：

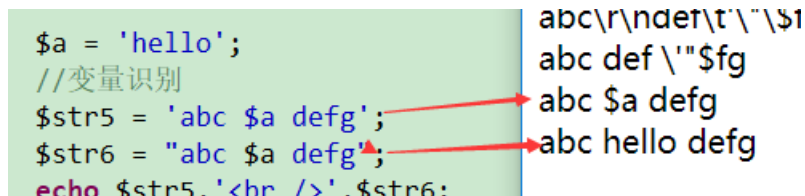
1、单引号字符串中能识别\'，双引号字符串中就不能识别\'。

```
//定义字符串识别转义符号
$str3 = 'abc\r\ndef\t\'\'\'\'$fg';
$str4 = "abc\r\ndef\t\'\'\'\'$fg";
echo $str3, '<br />', $str4;
```



2、双引号中因为能够识别\$符号，所以双引号中可以解析变量，单引号中不可以。

```
$a = 'hello';
//变量识别
$str5 = 'abc $a defg';
$str6 = "abc $a defg";
echo $str5, '<br />', $str6;
```



双引号中变量识别的规则：

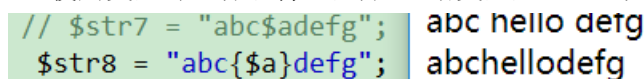
1) 变量本身系统能够与后面的内容区分：应该保证变量的独立性，不要让系统难以区分

```
$str7 = "abc$adefg";
```

Notice: Undefined variable: adefg in D:\Server\Apache24\htdocs\PHPBasic\string.php on line 24

2) 使用变量专业标识符（区分），给变量加上一组大括号 {}。

```
// $str7 = "abc$adefg";
$str8 = "abc{$a}defg";
```



字符串长度

1) 基本函数 strlen():得到字符串的长度（字节为单位）

2) 多字节字符串的长度问题：包含中文的长度

利用多字节字符串扩展模块：mbstring 扩展（mb: Multi Bytes）

首先需要加载 PHP 的 mbstring 扩展

```
;extension=interbase
;extension=ldap
;extension=mbstring
;extension=exif ; Must be after
extension=mysql
```

默认关闭，开启即可

可以使用 mb 扩展带来很多函数

mbstring 扩展针对的是一些关于字符统计：strlen()只是针对标准交换码 ASCII，mbstring 会针对不同的字符集。

如：mb_strlen(\$str,'utf-8');

字符串相关函数

1) 转换函数：implode()、explode()、str_split()

implode(连接方式，数组)：将数组中的元素按照某个规则连接成一个字符串

explode（分割字符，目标字符串）：将字符串按照某个格式进行分割，变成数组

str_split(字符串，字符长度)：按照指定长度拆分字符串得到数组

2) 截取函数: trim()、ltrim()、rtrim()

trim(字符串[,指定字符]): 本身默认用来去除两边的空格(中间不行), 但是也可以指定要去除的内容, 是按照指定的内容循环去除两边有的内容, 直到碰到一个不是目标字符串为止

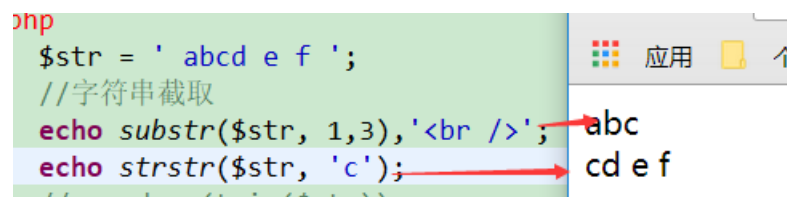
ltrim(): 去除左边的空格或者其他字符

rtrim(): 去除右边的空格或者其他字符

3) 截取函数: substr()、strstr()

substr(字符串, 起始位置从 0 开始[,长度]): 指定位置开始截取字符串, 可以截取指定长度(不指定到最后)

strstr(字符串, 匹配字符): 从指定位置开始截取到最后(可以用来取文件后缀名)



```
php
$str = ' abcd e f ';
//字符串截取
echo substr($str, 1, 3), '<br />';
echo strstr($str, 'c');
```

abc
cd e f

4) 大小转化函数: strtolower()、strtoupper()、ucfirst()

strtolower(字符串): 全部小写

strtoupper(字符串): 全部大写

ucfirst(字符串): 首字母大写(首字母为空格则看不出效果)

5) 查找函数: strpos()、strrpos()

strpos(): 判断字符在目标字符串中首次出现的位置

strrpos(): 判断字符在目标字符串中最后出现的位置

6) 替换函数: str_replace()

str_replace(mixed \$search, mixed \$replace, mixed \$subject): 将目标字符串中部分字符串进行替换

7) 格式化函数: printf()、sprintf()

printf()/sprintf(): (输出字符串有占位符, 顺序占位内容) 格式化输出数据

8) 其他: str_repeat()、str_shuffle()

str_repeat((string \$input, int \$multiplier): 重复某个字符串 n 次

str_shuffle(): 随机打乱字符串

数组

数组：array，数据的组合，指将一组数据（多个）存储到一个指定的容器中，用变量指向该容器，然后通过变量一次性得到该容器中的所有数据。

定义语法

PHP 中系统提供了多种定义数组的方式：

1、使用 array 关键字：最常用的

\$变量 = array(元素 1, 元素 2, 元素 3...);

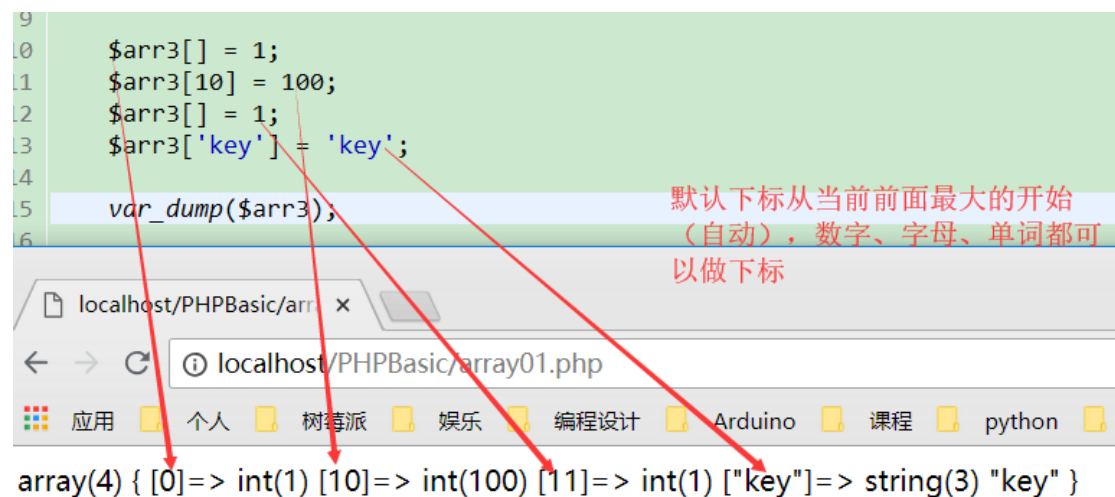
2、可以使用中括号来包裹数据：

\$变量 = [元素 1, 元素 2, 元素 3...];

3、隐形定义数组：给变量增加一个中括号，系统自动变成数组

\$变量[] = 值 1; //如果不提供下标，系统自动生成（数字:从 0 开始）

\$变量[下标] = 值; //中括号里面的内容称之为下标 key，该下标可以是字母（单词）或者数字，与变量命名规则相似



PHP 数组特点

1) 可以整数下标或者字符串下标

如果数组下标都为数字：索引数组

如果数组下标都为字符串：关联数组

2) 不同下标可以混合存在：混合数组

3) 数组元素的顺序以放入顺序为准，与下标无关

4) 数字下标的自增长特性：从 0 开始自动增长，如果中间手动出现较大的，那么后面的自增长元素从最大的值+1 开始

5) 特殊值下标的自动转换

布尔值：true/false

空类型：NULL

```
$arr4[false] = false;
$arr4[true] = true;
$arr4[NULL] = NULL;

var_dump($arr4);
```

array(3) { [0]=> bool(false) [1]=> bool(true) [""]=> NULL }

不要用特殊值做下标

6) PHP 中数组元素没有类型限制

7) PHP 中数组元素没有长度限制

补充: PHP 中数组是很大的数据, 存储的位置是堆区, 为当前数组分配一块连续的内存

多维数组

多维数组: 数组里面的元素又是数组

二维数组:

```
19 $arr4[NULL] = NULL;
20
21 // var_dump($arr4);
22
23 $info = array(
24     array('name' => 'jim', 'age' => 30),
25     array('name' => 'tom', 'age' => 28),
26     array('name' => 'lily', 'age' => 20)
27 );
28 echo '<pre>';
29 print_r($info);
```

key—value

最后一个元素可以跟逗号(不建议)

```
Array
(
    [0] => Array
        (
            [name] => jim
            [age] => 30
        )
    [1] => Array
        (
            [name] => tom
            [age] => 28
        )
    [2] => Array
        (
            [name] => lily
            [age] => 20
        )
)
```

多维数组:

在第二维的数组元素中可以继续是数组, 在 PHP 中, 没有维度的限制。

但是不建议使用超过三维以上的数组, 会增加访问的复杂度, 降低访问效率。

异形数组 (不规则数组):

异形数组: 数组中的元素不规则, 有普通基本变量也有数组。

在实际开发中并不常用, 尽量让数组元素规则化, 便于进行访问。

遍历数组

数组遍历: 普通数组数据的访问都是通过数组元素的下标进行访问, 如果说数组中所有的数组都需要依次输出数来, 就需要我们使用到一些简化的规则来实现自动获取下标以及输出数组元素。

```
$arr = array(0 => array('name' => 'Tom'), 1 => array('name' => 'Jim')); //二维数组
```

```
//访问一维元素: $arr[一维下标]
```

```
$arr[0]; //结果: array('name' => 'Tom');
```

```
//访问二维元素: $arr[一维下标][二维下标]
```

```
$arr[1]['name']; //结果: Jim
```

foreach 遍历语法

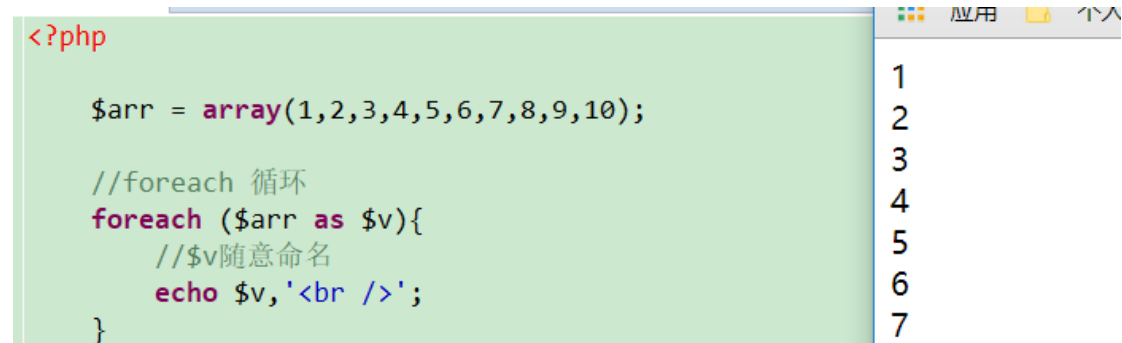
基本语法如下:

```
foreach ($数组变量 as [$下标 =>] $值) {  
    //通过$下标访问元素的下标; 通过$值访问元素的值  
}
```

通常: 如果是关联数组 (字母下标), 就需要下标。如果是数字下标, 就直接访问值。

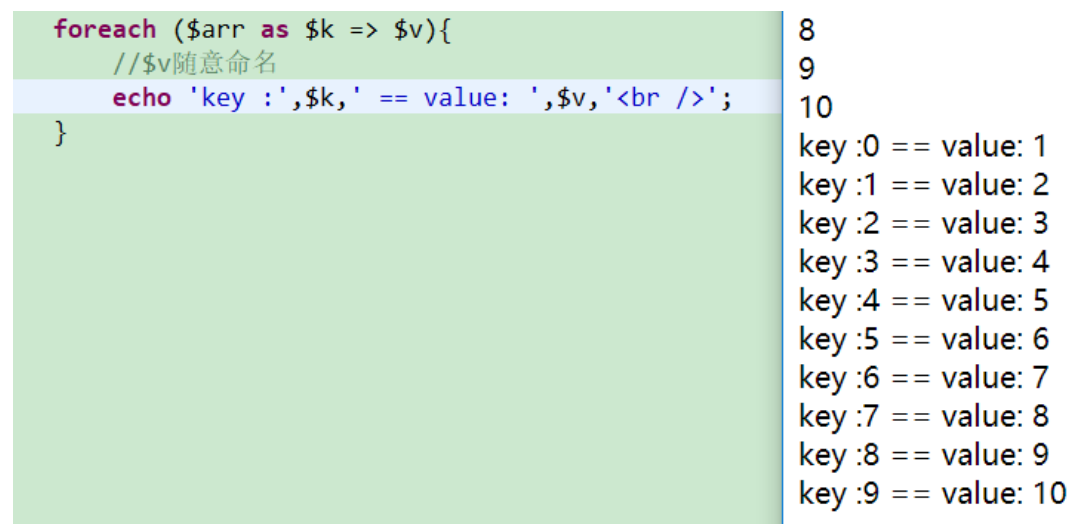
直接访问:

```
<?php  
  
$arr = array(1,2,3,4,5,6,7,8,9,10);  
  
//foreach 循环  
foreach ($arr as $v){  
    // $v 随意命名  
    echo $v, '<br />';  
}
```



获取下标:

```
foreach ($arr as $k => $v){  
    // $v 随意命名  
    echo 'key :', $k, ' == value: ', $v, '<br />';  
}
```



通常在进行数据存储定义的时候, 通常二维数组不会两个维度的 key 下标都为数字, 一般是一维为数字 (无意义), 二维为字符串 (数据库表字段), 所以在进行遍历的时候, 通常是只需要针对一维进行遍历, 取得二维数组元素, 然后二维数组通过下标去访问。

foreach 遍历原理

foreach 遍历原理:本质是数组的内部有一颗指针,默认是指向数组元素的第一个元素,foreach 就是利用指针去获取数据,同时移动指针。

```
foreach ($arr as $v) {  
    //循环体  
}
```

- 1、foreach 会重置指针: 让指针指向第一个元素;
- 2、进入 foreach 循环: 通过指针取得当前第一个元素,然后将下标取出放到对应的下标变量中(如果存在),将值取出来放到对应的值变量\$*v* 中;(指针下移)
- 3、进入到循环内部(循环体),开始执行;
- 4、重复 2 和 3 直到在 2 的时候遇到指针去不到内容(指针指向数组最后)

for 循环遍历数组

for 循环: 基于已知边界条件(起始和结束)然后有条件的变化(规律)

因此, for 循环遍历数组有对应条件

- 1、获取数组长度: `count(数组)` 得到数组元素的长度
- 2、要求数组元素的下标是规律的数字(起始和变化规律)

while 配合 each 和 list 遍历数组

each 函数的使用: each 能够从一个数组中获取当前数组指针指向的元素的`下标`和`值`,拿到之后将数组指针下移,同时将拿到的元素下标和值一个四个元素的数组返回:

0 下标 => 取得元素的下标值

1 下标 => 取得元素的值

key 下标 => 取得的元素的下标值

value 下标 => 取得元素的值

不赞成使用

Deprecated: The `each()` function is deprecated.

The screenshot shows a PHP code snippet in a green-themed editor. The code is as follows:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

The code snippet is:

```
*/  
$arr = array(1, 'name' => 'Tom', 'age' => 30);  
  
echo '<pre>';  
//each函数指针操作  
print_r(each($arr));  
print_r(each($arr));
```

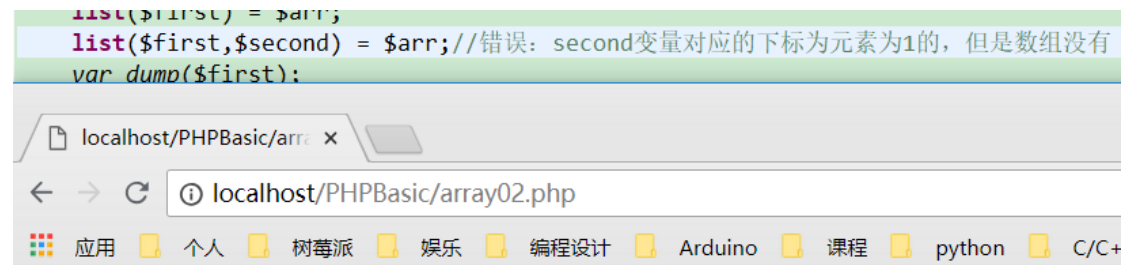
Red arrows point from the `each($arr)` calls to the right, where the internal structure of the returned array is shown:

```
Array  
(  
    [1] => 1  
    [value] => 1  
    [0] => 0  
    [key] => 0  
)  
  
Array  
(  
    [1] => Tom  
    [value] => Tom  
    [0] => name  
    [key] => name  
)
```


list 函数的使用：list 是一种结构，不是一种函数（没有返回值），是 list 提供一堆变量去从一个数组中取得元素值，然后依次存放到对应的变量当中（批量为变量赋值，值来源于数组）：

list 必须从索引数组中去获取数据，而且必须从 0 开始。

错误操作：变量多余数组元素，没有指定从 0 到指定变量的下标的数组元素。



Notice: Undefined offset: 1 in D:\Server\Apache24\htdocs\PHPBasic\array02.php on line 25

list 与 each 配合特别好：each 一定有两个元素就是 0 和 1 下标元素。

list（变量 1，变量 2）= each(数组); // 是一种赋值运算，但是可以得到 false 结果，（each 得不到正确的结果），整个表达式为 false。（false 代表数组到头了）

```
*/  
$arr =array(1,'name' => 'Tom',3,'age' => 30,5,6,7,8);  
while ( list($key,$value) = each($arr)){  
    //list搭配each  
    //输出  
    echo 'key is :'.$key . ' value is:' . $value.'<br />';  
}  
/ echo '<pre>';  
//each函数指针操作  
// print_r(each($arr));
```

```
key is :0 value is:1  
key is :name value is:Tom  
key is :1 value is:3  
key is :age value is:30  
key is :2 value is:5  
key is :3 value is:6  
key is :4 value is:7  
key is :5 value is:8
```

数组的相关函数

1) 排序函数:对数组元素进行排序

sort():顺序排序（下标重排）

rsort():逆序排序

asort():顺序排序（下标保留）

arsort(): 逆序排序

ksort():顺序排序（按照键名（下标））

krsort():逆序排序

shuffle():随机打乱数组元素

<pre> 3 4 \$arr = array(3,1,5,2,0); 5 echo '<pre>'; 6 7 //sort(\$arr); 8 asort(\$arr); 9 print_r(\$arr); </pre>	<pre> Array ([4] => 0 [1] => 1 [3] => 2 [0] => 3 [2] => 5) </pre>
---	--

2) 指针函数：实现自定义循环

reset(): 重置指针，将数组指针回到首位，取得第一个元素的值

end(): 重置指针，将数组指针指到最后一个元素，取得最后一个元素的值

next(): 指针下移，取得下一个元素的值

prev(): 指针上移，取得上一个元素的值

current(): 获取当前指针对应的元素值

key(): 获取当前指针对应的下标值

注意事项: NEXT 和 PREV 会移动指针，有可能导致指针移动到最前或者最后（离开数组），导致数组不能使用，通过 NEXT 和 PREV 不能回到正确的指针位置，只能通过 END 或者 RESET 进行指针重置。

3) 其他函数

count(): 统计数组中元素的数量

array_push(): 往数组中加入一个元素（数组后面）

array_pop(): 从数组中取出一个元素（数组后面）

array_shift(): 从数组中取出一个元素（数组前面）

array_unshift(): 从数组中压入一个元素（数组前面）

array_reverse(): 数组元素反过来

in_array(): 判断一个元素在数组中是否存在

array_keys(): 获取一个数组的所有下标，返回一个索引数组

array_values(): 获取一个数组的所有值，返回一个索引数组

array_slice(array \$array, int \$offset): 函数在数组中根据条件取出一段值，并返回。