目录

[PHP基础 2](#_Toc522960766)

[PHP代码标记 2](#_Toc522960767)

[PHP注释 2](#_Toc522960768)

[PHP语句分隔符 2](#_Toc522960769)

[PHP变量 3](#_Toc522960770)

[常量 3](#_Toc522960771)

[数据类型 4](#_Toc522960772)

[流程控制 6](#_Toc522960773)

[文件 7](#_Toc522960774)

[函数 9](#_Toc522960775)

[闭包 11](#_Toc522960776)

[常用系统函数 12](#_Toc522960777)

[错误处理 13](#_Toc522960778)

[错误显示设置 15](#_Toc522960779)

[字符串类型 17](#_Toc522960780)

[字符串相关函数 19](#_Toc522960781)

[数组 21](#_Toc522960782)

[遍历数组 22](#_Toc522960783)

[数组的相关函数 25](#_Toc522960784)

# **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:整形大小。

PHP\_INT\_MAX:整形能表示的最大值（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：
   1. 以字母开头的字符串，永远为0。
   2. 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。

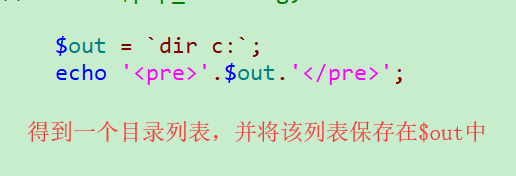
错误抑制符：有一些错误提前可以预知，又无法避免，但又不希望给用户看，可以是用错误抑制符。

@：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。



## 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

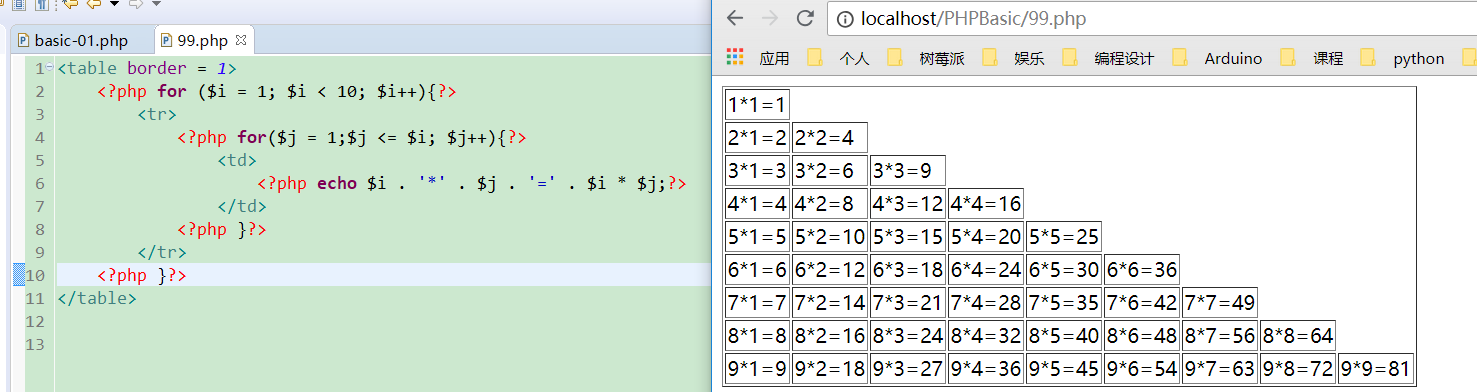
终止控制：循环直接进结束。

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。

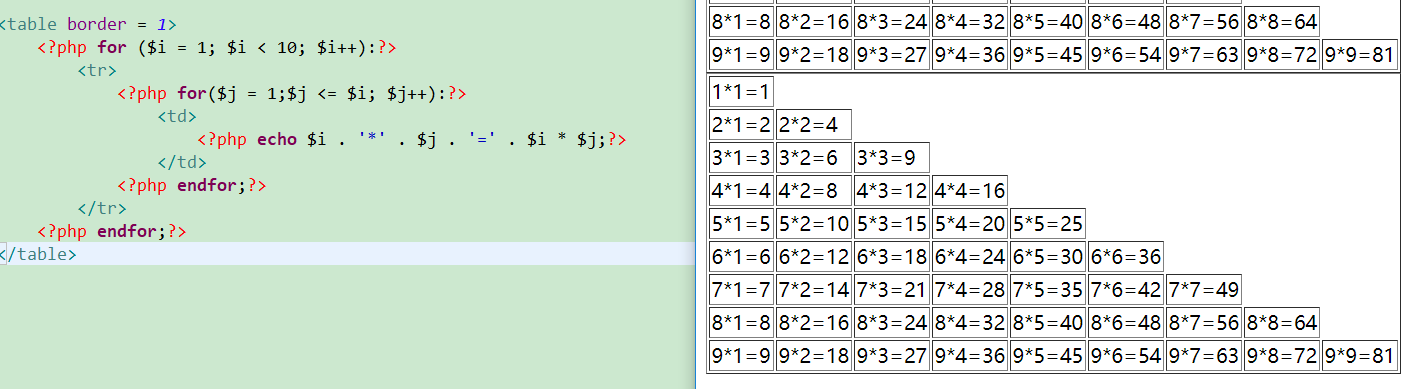
PHP与HTML混搭。



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

1. 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：知道脚本运行结束（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’；

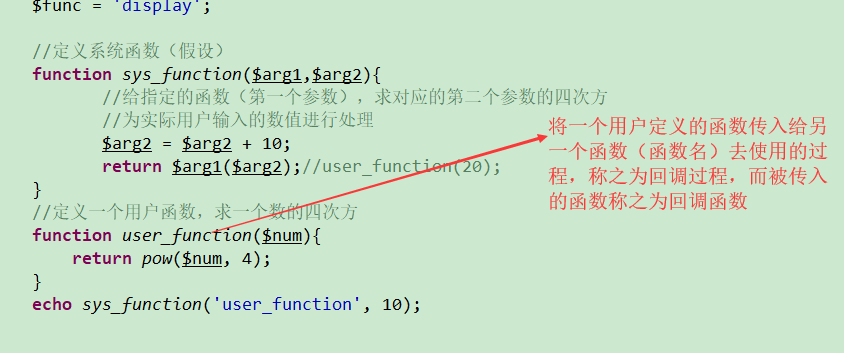
function display(){

}

//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，需要用户在外部定义一个自定义函数，但是是需要传入到系统函数内部使用。



**匿名函数**

匿名函数：没有名字的函数。

基本语法：

变量名 = function(){

}；

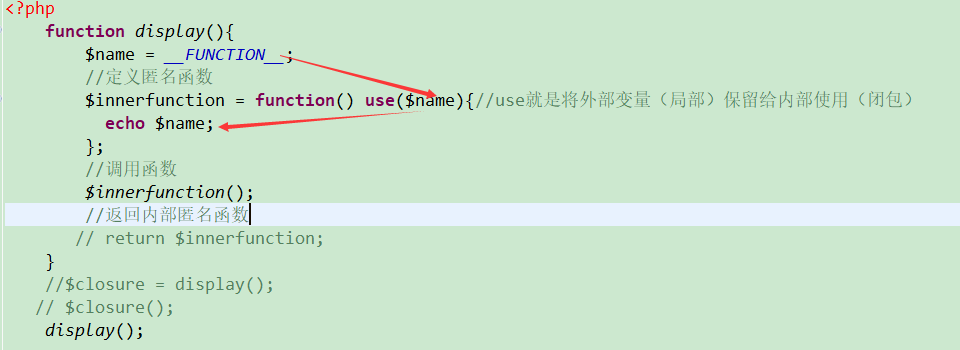
变量名();//调用

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）。



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）

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。

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

printf()：格式化输出

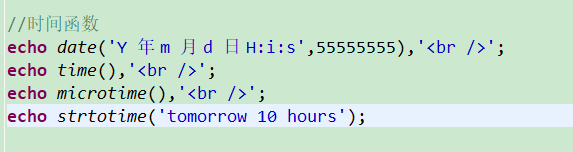
**2）有关时间的函数**

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。



**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

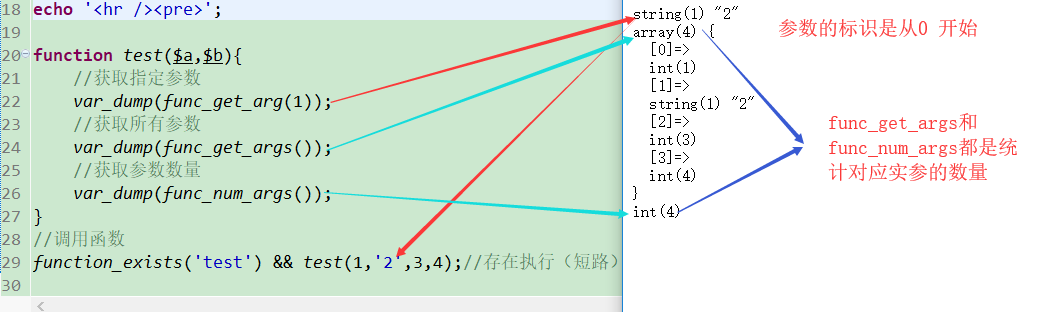
**4）有关函数的函数**

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



## 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

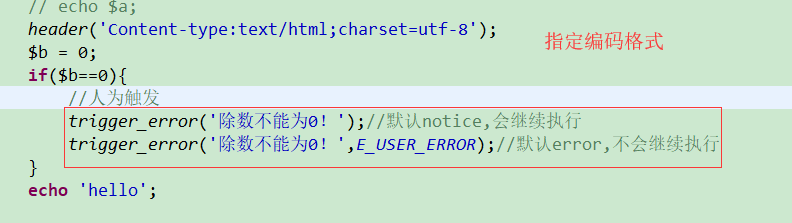
只要警告和通知：E\_WARNING | E\_NOTICE

**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。



## 错误显示设置

错误显示设置：哪些错误该显示，以及该如何显示

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





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**

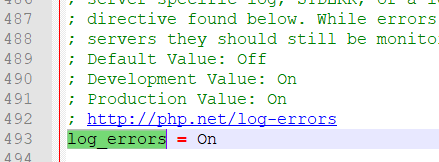
在实际生产环境中，不会直接让错误赤裸裸的展示给用户：

1、不友好

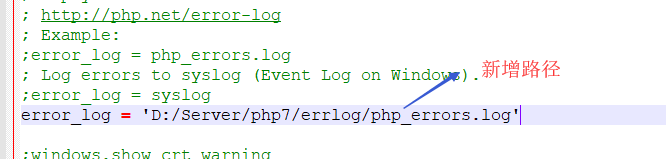
2、不安全，错误会暴露网站很多信息（路径、文件名）

但错误不可能避免，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



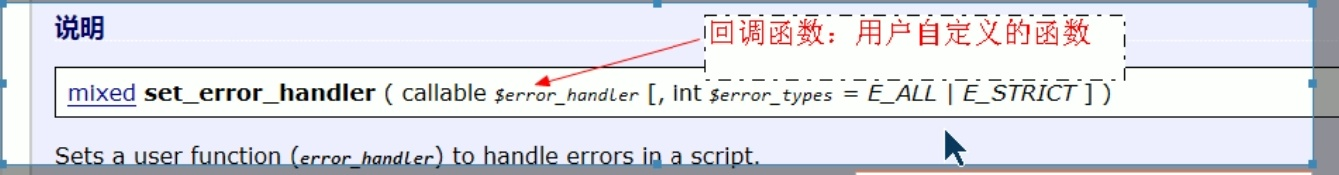
2、指定路径



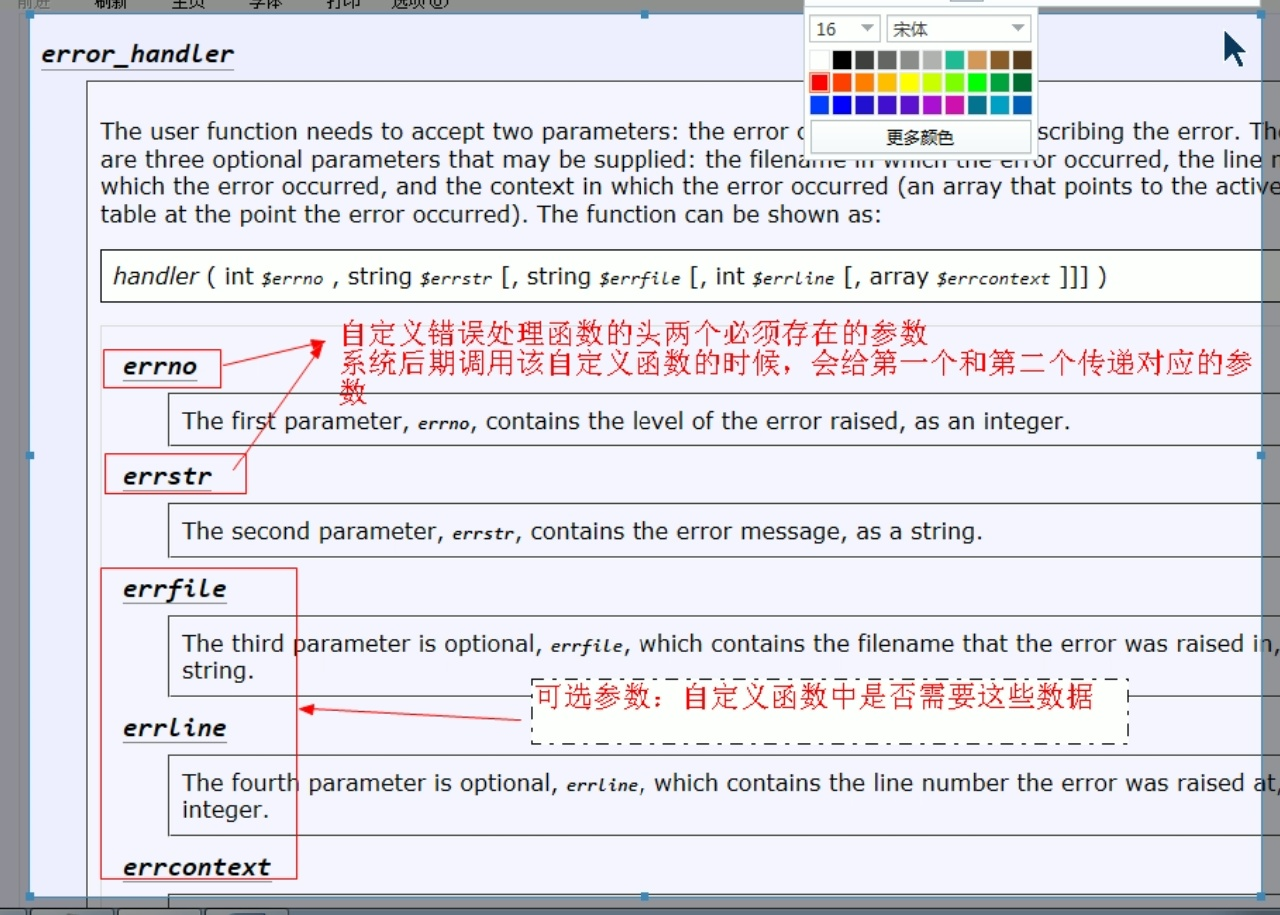
**自定义错误处理**

最简单的错误处理：trigger\_errors()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。

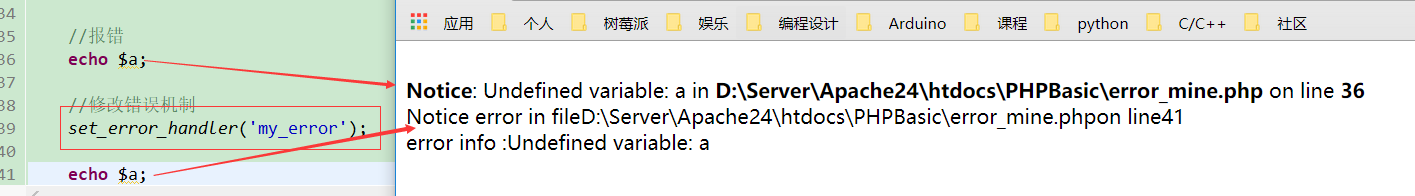


代码实现：

1、自定义错误处理函数：注意参数



2、注册自定义函数：修改错误处理机制



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

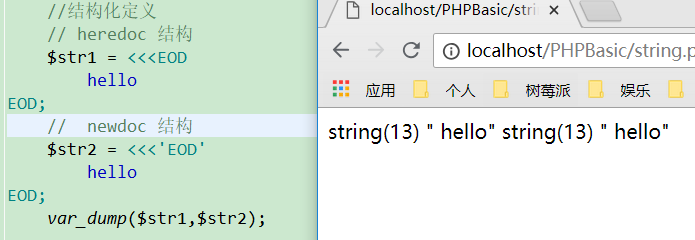
边界符;

4）heredoc字符串：没有双引号的双引号字符串

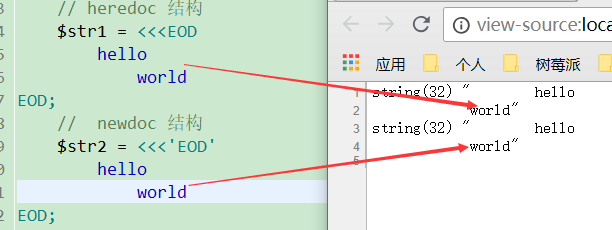
$str = <<<边界符

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

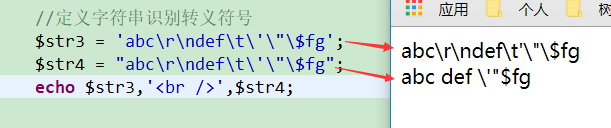
\” ：在双引号字符串中显示双引号

\$ ：在PHP中使用$符号作为变量符号，因此需要特别识别。

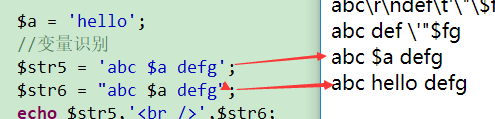
\r \n \t

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\’。



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。



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



2）使用变量专业标识符（区分），给变量加上一组大括号{}。



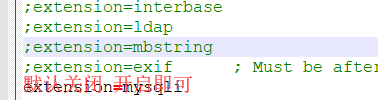
**字符串长度**

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



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

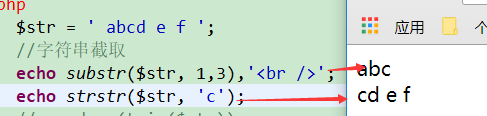
ltrim():去除左边的空格或者其他字符

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）



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，然后可以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…);

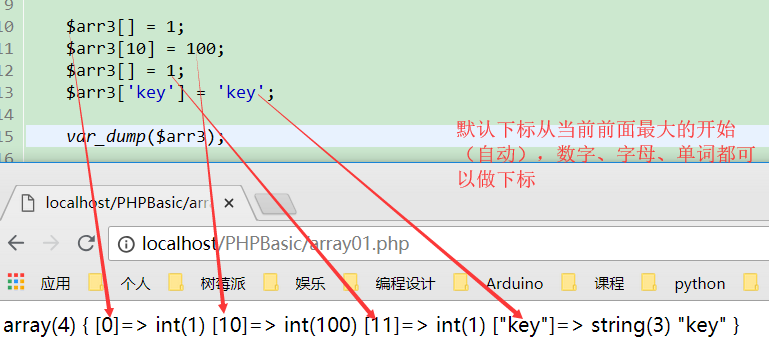
2、可以使用中括号来包裹数据：

$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

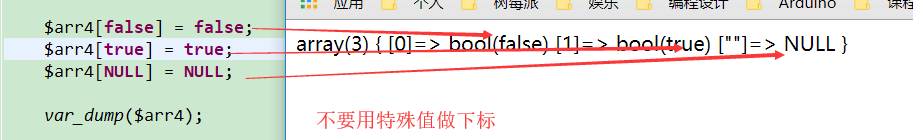
3）数组元素的顺序以放入顺序为准，与下标无关

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



6）PHP中数组元素没有类型限制

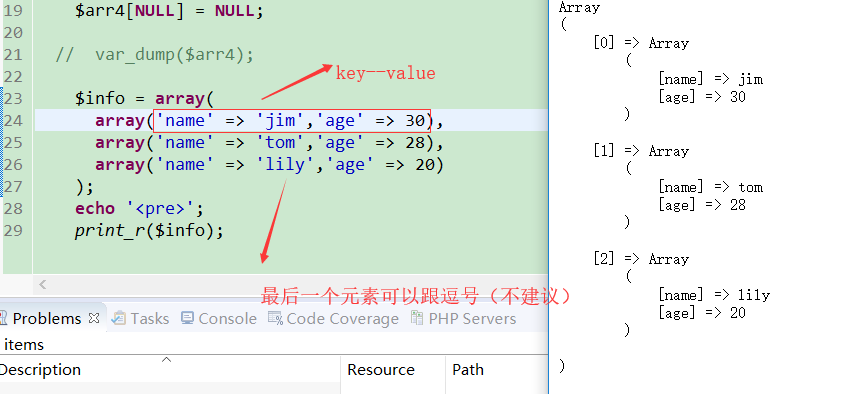
7）PHP中数组元素没有长度限制

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：



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][‘nam’]; //结果：Jim

**foreach遍历语法**

基本语法如下：

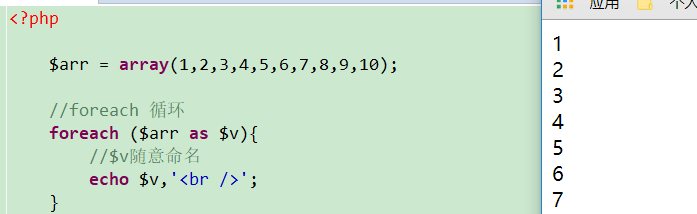
foreach（$数组变量 as [$下标 => ] $值）{

//通过$下标访问元素的下标；通过$值访问元素的值

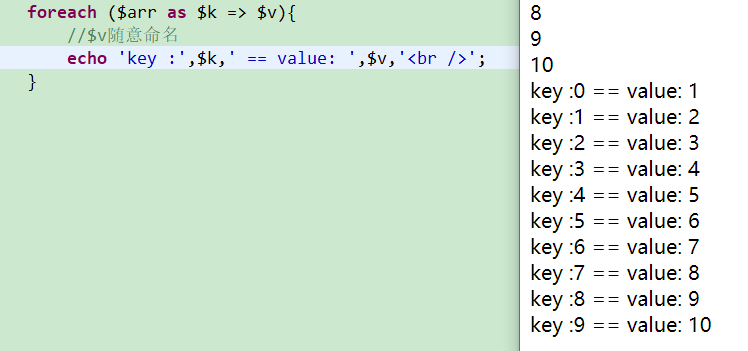
}

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：



获取下标：



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循环遍历数组由对应条件

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**

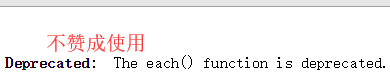
each函数的使用：each 能够从一个数组中获取当前数组指针指向的元素的下标和值，拿到之后将数组指针下移，同时将拿到的元素下标和值一个四个元素的数组返回：

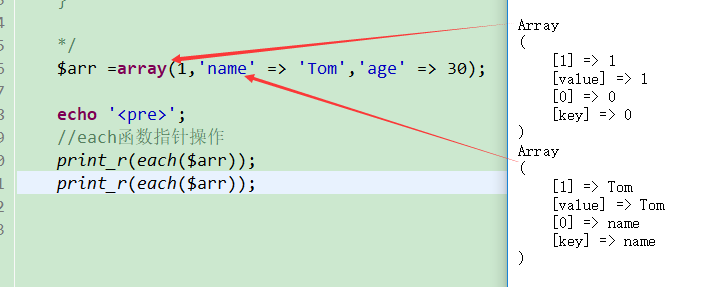
0 下标 =》取得元素的下标值

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

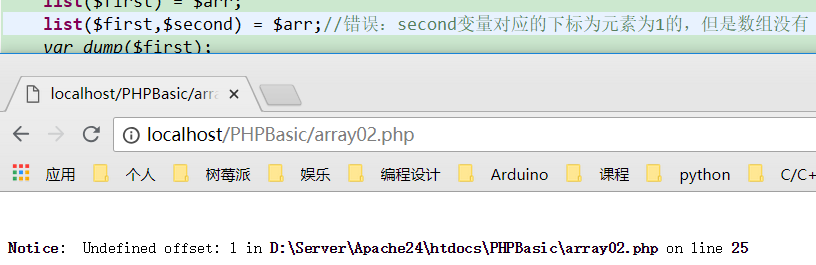




list函数的使用：list是一种结构，不是一种函数（没有返回值），是list提供一堆变量去从一个数组中取得元素值，然后依次存放到对应的变量当中（批量为变量赋值，值来源于数组）：

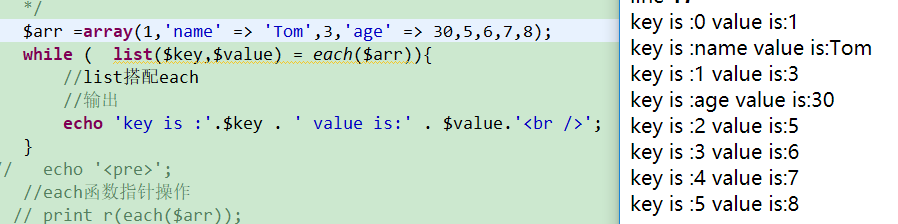
list必须从索引数组中去获取数据，而且必须从0开始。

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。



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去不到正确的结果），整个表达式为false。（false代表数组到尽头了）



## 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

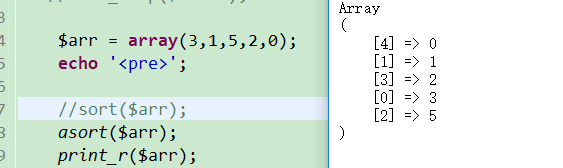
asort():顺序排序（下标保留）

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



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():从数组中取出一个数组（数组后面）

array\_shift()：从数组中取出一个元素（数组前面）

array\_unshift():从数组中压入一个函数（数组前面）

array\_reverse():数组元素反过来

in\_arrray():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，返货一个索引数组

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。