

常用系统函

1) 有关输出的函数

`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)

```
//系统函数
```

```
//输出相关
```

```
echo print('hello world<br/>');
```

```
print 'hello world<br/>';
```

```
$a = 'hello world<br/>';
```

```
print_r($a);
```



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

B	---	---
d	月份中的第几天，有前导零的 2 位数字	01 到 31
D	星期中的第几天，文本表示，3 个字母	Mon 到 Sun
j	月份中的第几天，没有前导零	1 到 31
l ("L"的小写字母)	星期几，完整的文本格式	Sunday 到 Saturday
N	ISO-8601 格式数字表示的星期中的第几天 (PHP 5.1.0 新加)	1 (表示星期一) 到 7 (表示星期天)
S	每月天数后面的英文后缀，2 个字符	st, nd, rd 或者 th 。可以和 j 一起用
w	星期中的第几天，数字表示	0 (表示星期天) 到 6 (表示星期六)
z	年份中的第几天	0 到 366
星期	---	---
W	ISO-8601 格式年份中的第几周，每周从星期一开始 (PHP 4.1.0 新加的)	例如： 42 (当年的第 42 周)
月	---	---
F	月份，完整的文本格式，例如 January 或者 March	January 到 December
m	数字表示的月份，有前导零	01 到 12
M	三个字母缩写表示的月份	Jan 到 Dec
n	数字表示的月份，没有前导零	1 到 12
t	给定月份所应有的天数	28 到 31
年	---	---
L	是否为闰年	如果是闰年为 1 ，否则为 0

time(): 获取当前时间对应的的时间戳

microtime(): 获取微秒级别的时间

```

13
14 //时间函数
15 echo date('Y 年 m 月 d 日 H:i:s',12345678), '<br/>';
16 echo time(), '<br/>';
17 echo microtime(), '<br/>';

```

```

hello world
hello world
hello world
1970 年 05 月 24 日 05:21:18
1494913599
0.76953200 1494913599

```

Strtotime(): 按照规定格式的字符串转换成时间戳

```

echo time(), '<br/>';
echo microtime(), '<br/>';

echo strtotime('tomorrow 10 hours');

```

```

hello world
hello world
1970 年 05 月 24 日 05:21:18
1494913701
0.46972600 1494913701
1494986400

```

3) 有关数学的函数

max(): 指定参数中最大的值

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(): 求指定数字的指定指数次结果: pow(2,8) == 2^8 == 256

abs(): 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，让代码安全性更高）

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(): 在自定义函数中获取所有的参数（数组）

func_num_args(): 获取当前自定义函数的参数数量

```
//函数相关函数
function test($a,$b){
    //获取指定参数
    var_dump(func_get_arg(1));

    //获取所有参数
    var_dump(func_get_args());

    //获取参数数量
    var_dump(func_num_args());
}

//调用函数
function_exists('test') && test(1,'2',3,4);
```

1hello world
hello world

1970 年 05 月 24 日 05:21:18
1494914310
0.41015600 1494914310
1494986400

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都是统计的对应实参的数量

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 的语法规则，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，但是代码在执行的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，但是得不到想要的结果

```
$a = 10;
if($a = 1){ //最常见把比较符号写成赋值符号
    //执行代码
}
```

错误代号

所有看到的错误代号在 PHP 中都被定义成了系统常量（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, 代表着所有从错误 (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

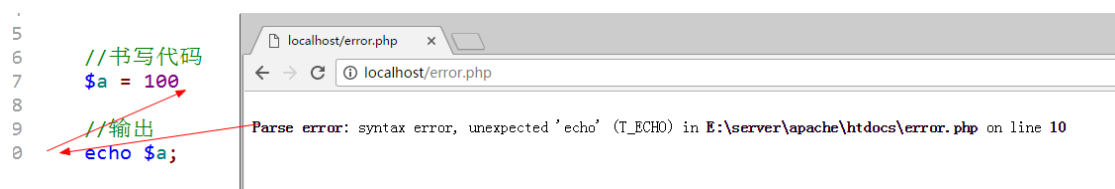
所有以 E 开头的错误常量 (代号) 其实都是由一个字节存储, 然后每一种错误占据一个对应的位, 如果想进行一些错误的控制, 可以使用位运算进行操作

排除通知级别 notice: E_ALL & ~E_NOTICE

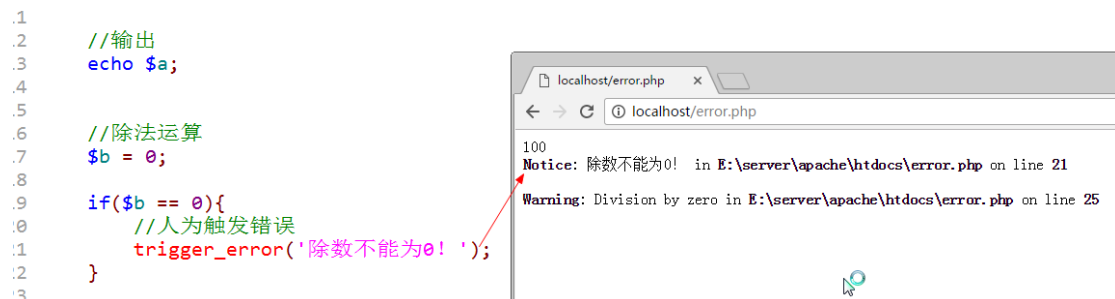
只要警告和通知: E_WARNING | E_NOTICE

错误触发

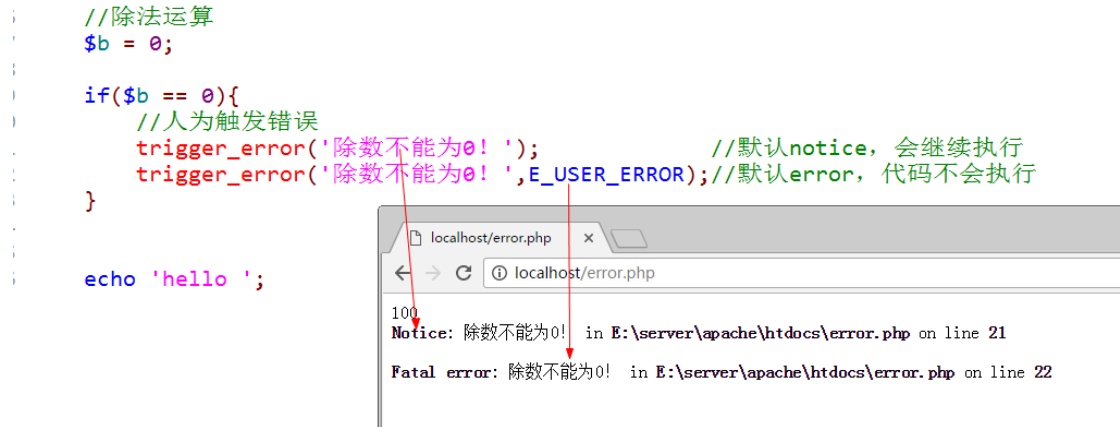
程序运行时触发: 系统自动根据错误发生后, 对比对应的错误信息, 输出给用户: 主要针对代码的语法错误和运行时错误。



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(错误提示):



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

```
460 ; http://php.net/error-reporting
461 error_reporting = E_ALL
462
463 ; This directive controls whether or not and where PHP will output errors,
464 ; notices and warnings too. Error output is very useful during development,
465 ; it could be very dangerous in production environments. Depending on the
466 ; which is triggering the error, sensitive information could potentially leak
467 ; out of your application such as database usernames and passwords or worse.
468 ; It's recommended that errors be logged on production servers rather than
469 ; having the errors sent to STDOUT.
470 ; Possible Values:
471 ;   Off = Do not display any errors
472 ;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
473 ;   On or stdout = Display errors to STDOUT
474 ; Default Value: On
475 ; Development Value: On
476 ; Production Value: Off
477 ; http://php.net/display-errors
478 display_errors = On
479
```

2、可以在运行的 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_reporting',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）

所以在生产环境中，一般不显示错误（错误也比较少），但是不可能避免会出现错误（测试的时候不会发现所有的问题），这个时候不希望看到，但是又希望捕捉到可以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 `error_log` 配置项

- 1、开启日志功能

```
495 ; Default Value: Off
496 ; Development Value: On
497 ; Production Value: On
498 ; http://php.net/log-errors
499 log_errors = On
500
```

- 2、指定路径

```
~ , Example:
14 ;error_log = php_errors.log
15 ; Log errors to syslog (Event Log on NT, not valid in Windows 95)
16 ;error_log = syslog
17 error_log = 'E:/server/php5/errlog/php_errors.log'
18
19 ;windows.show_crt_warning
```

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，然后将该函数增加操作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()`

说明

回调函数：用户自定义的函数

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

Sets a user function (*error handler*) to handle errors in a script.

- 2、自定义错误处理函数，系统有要求

error_handler

The user function needs to accept two parameters: the error code, and a string describing the error. There are three optional parameters that may be supplied: the filename in which the error occurred, the line in which the error occurred, and the context in which the error occurred (an array that points to the active table at the point the error occurred). The function can be shown as:

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

errno

自定义错误处理函数的头两个必须存在的参数
系统后期调用该自定义函数的时候，会给第一个和第二个传递对应的参数

The first parameter, *errno*, contains the level of the error raised, as an integer.

errstr

The second parameter, *errstr*, contains the error message, as a string.

errfile

The third parameter is optional, *errfile*, which contains the filename that the error was raised in as a string.

可选参数：自定义函数中是否需要这些数据

errline

The fourth parameter is optional, *errline*, which contains the line number the error was raised at as an integer.

errcontext

代码实现：

1、自定义错误处理函数：注意参数

```
1 <?php
2
3 //自定义错误处理机制
4 header('Content-type:text/html;charset=utf-8');
5
6 //自定义函数
7 /*
8  @param1 $errno, 是系统提供的错误代码: E_ALL, E_NOTICE..
9  */
10 function my_error($errno,$errstr,$errfile,$errline){
11     //判断：当前会碰到错误有哪些
12     if(!(error_reporting() & $errno)){
13         return false;
14         //error_reporting没有参数代表获取当前系统错误处理对应的级别
15     }
16
17     //开始判断错误类型
18     switch($errno){
19         case E_ERROR:
20         case E_USER_ERROR:
21             echo 'fatal error in file ' . $errfile . ' on line ' . $errline . '<br/>';
22             echo 'error info : ' . $errstr;
23             break;
```

```

24         case E_WARNING:
25         case E_USER_WARNING:
26             echo 'Warning in file ' . $errfile . ' on line ' . $errline . '<br/>';
27             echo 'error info : ' . $errstr;
28             break;
29         case E_NOTICE:
30         case E_USER_NOTICE:
31             echo 'Notice in file ' . $errfile . ' on line ' . $errline . '<br/>';
32             echo 'error info : ' . $errstr;
33             break;
34     }
35
36     return true;
37 }
38
39 //报错
40 echo $a;
41
42 //修改错误机制
43 set_error_handler('my_error');
44
45 echo $a;

```

原来系统报错

修改错误机制后，使用自定义报错

2、注册自定义函数：修改错误处理机制

```

    return true;
}

//报错
echo $a;

//修改错误机制
set_error_handler('my_error');

echo $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

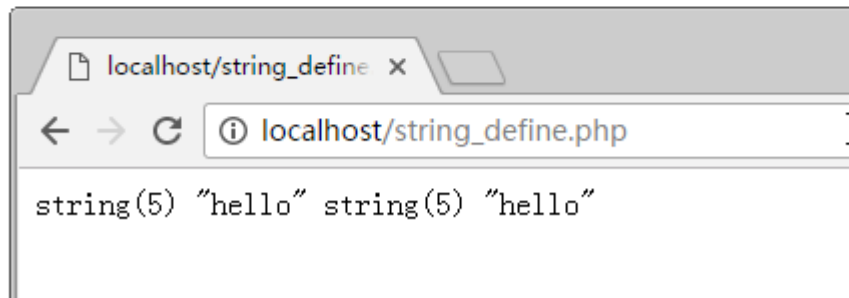
字符串定义语法

- 1) 单引号字符串：使用单引号包裹
- 2) 双引号字符串：使用双引号包裹


```

2
3 //PHP字符串：定义
4
5 //引号定义
6 $str1 = 'hello';
7 $str2 = "hello";
8 var_dump($str1,$str2);

```



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：没哟双引号的双引号字符串

\$str = <<<边界符

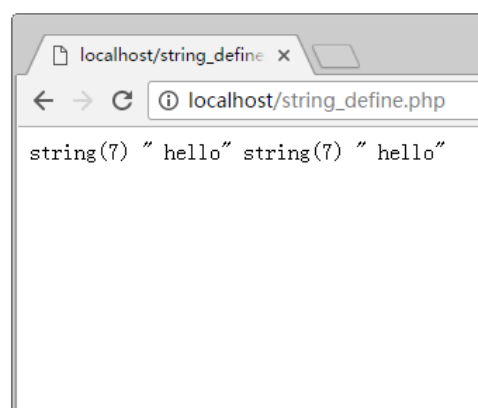
字符串内容

边界符;

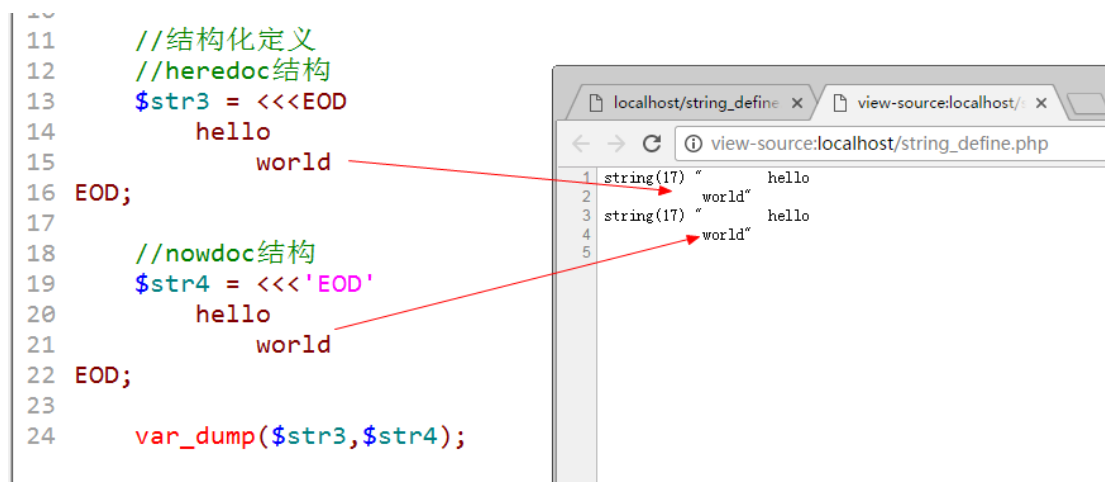
```

11 //结构化定义
12 //heredoc结构
13 $str3 = <<<EOD
14     hello
15 EOD;
16
17 //nowdoc结构
18 $str4 = <<<'EOD'
19     hello
20 EOD;
21
22 var_dump($str3,$str4);

```



Heredoc 和 nowdoc 比引号还是要区别多一点



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 在识别转义字符的时候也是使用同样的模式：反斜杠+字母

在 PHP 中系统常用的转义符号：

\\：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

\r：代表回车（理论上是回到当前行的首位置）

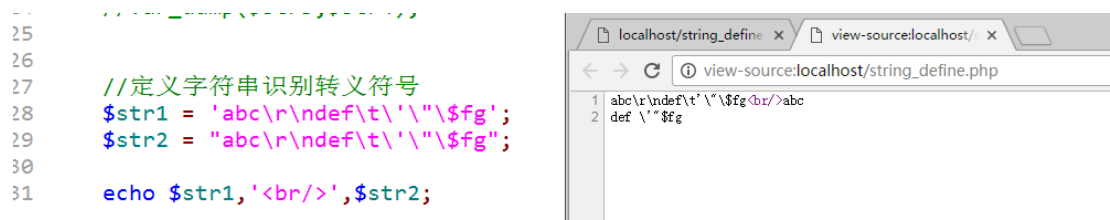
\n：代表新一行

\t：类似 tab 键，输出 4 个空格

\\\$：在 PHP 中使用 \$ 符号作为变量符号，因此需要特定识别

单引号和双引号的区别：

1、其中单引号中能够识别\\，而双引号中就不能识别\\（下图浏览器查看的是页面源代码）

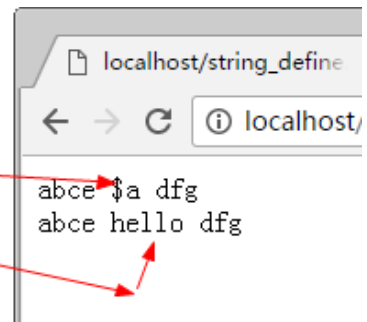


2、双引号中因为能够识别\$符号，所以双引号中可以解析变量，而单引号不可以

```

$a = 'hello';
//变量识别
$str1 = 'abce $a dfg';
$str2 = "abce $a dfg";
echo $str1, '<br/>', $str2;

```



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

```

$a = 'hello';
//变量识别
$str1 = 'abce $a dfg';
$str2 = "abce $a dfg";
$str3 = "abce$adfg";
echo $str1, '<br/>', $str2, $str3;

```

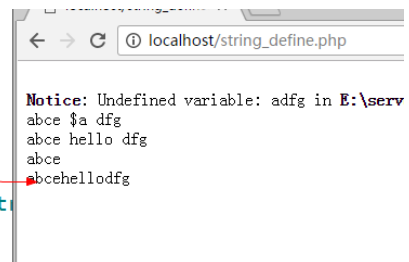


- 2) 使用变量专业标识符（区分），给变量加上一组大括号{}

```

$a = 'hello';
//变量识别
$str1 = 'abce $a dfg';
$str2 = "abce $a dfg";
$str3 = "abce$adfg";
$str4 = "abce{$a}dfg";
echo $str1, '<br/>', $str2, '<br/>', $str3, '<br/>', $str4;

```



结构化定义字符串变量的规则：

- 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；
 - 1.2 下边界符必须顶格：最左边；
 - 1.3 下边界同样后面只能跟分号，不能跟任何内容；
- 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

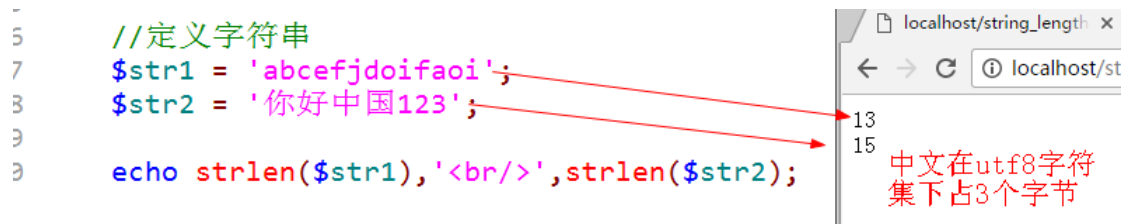
```

1  $str1 = "abce $a dfg";
2  echo $str1, '<br/>', $str2, '<br/>', $str3;
3
4  $str1 = <<<EOD
5  //这是弹出内容
6  <script>
7      alert('$str1'); //js弹出字符串
8  </script>
9  EOD;
10 echo $str1;

```

字符串长度问题

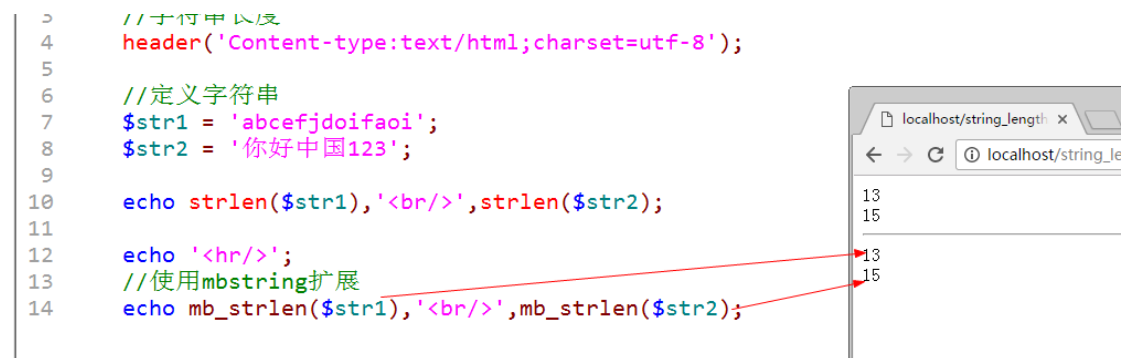
- 1) 基本函数 strlen()：得到字符串的长度（字节为单位）



- 2) 多字节字符串的长度问题: 包含中文的长度
 - 3) 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

```
379 ;extension=php_ldap.dll
380 extension=php_mbstring.dll
381 ;extension=php_exif.dll      ; Must be after mbstring as of 5.3.0
```

可以使用 mb 扩展带来很多函数

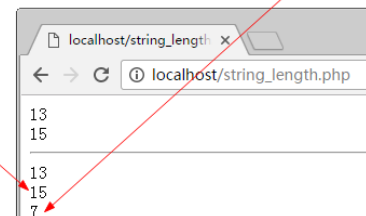


Mbstring 扩展针对的是一些关于字符统计: `strlen` 只是针对标准交换码 ASCII, `mbstring` 会针对不同的字符集

```
$str1 = 'abcefjdoifaai';
$str2 = '你好中国123';

echo strlen($str1), '<br/>', strlen($str2);

echo '<hr/>';
//使用mbstring扩展
echo mb_strlen($str1), '<br/>', mb_strlen($str2), '<br/>', mb_strlen($str2, '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

中国|北京|顺义 == array('中国','北京','顺义');

Str_split(字符串,字符长度): 按照指定长度拆分字符串得到数组

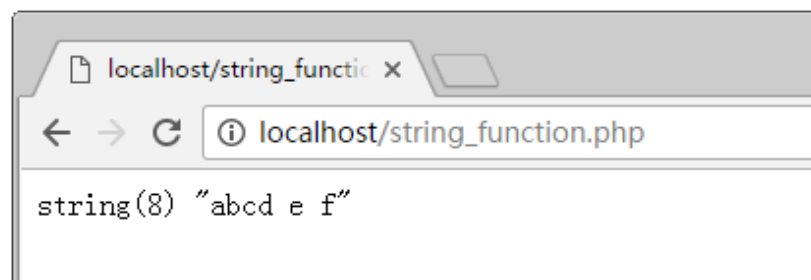
2) 截取函数: trim(), ltrim(), rtrim()

Trim(字符串[,指定字符]): 本身默认是用来去除字符串两边的空格(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为止

Ltrim(): 去除左边的

Rtrim(): 去除右边的

```
1  
2  
3     $str = ' abcd e f ';  
4  
5  
6  
7     var_dump(trim($str));
```

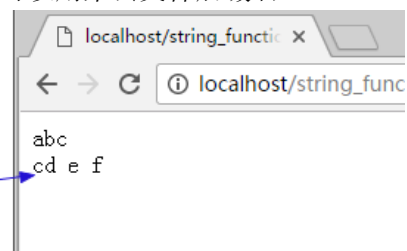


3) 截取函数: substr(), strstr()

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(可以用来去文件后缀名)

```
1  
2  
3     $str = ' abcd e f ';  
4  
5  
6     //字符串截取  
7     echo substr($str,1,3), '<br/>';  
8     echo strstr($str, 'c');
```



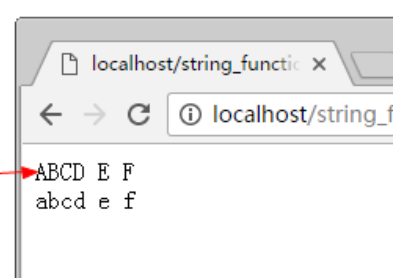
4) 大小转换函数: strtolower(), strtoupper(), ucfirst()

Strtolower: 全部小写

Strtoupper: 全部大写

Ucfirst: 首字母大写

```
1  
2  
3  
4  
5     $str = ' abcd e f ';  
6  
7     //字符串大小写  
8     echo strtoupper($str), '<br/>';  
9     echo ucfirst($str);  
10
```

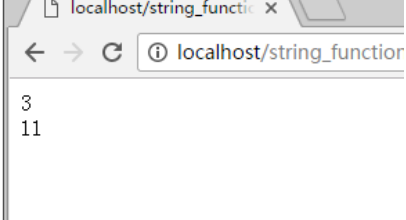


5) 查找函数: strpos(), strrpos()

Strpos(字符串, 匹配字符): 判断字符在目标字符串中出现的位置 (首次)

Strrpos(字符串, 匹配字符): 判断字符在目标字符串中最后出现的位置

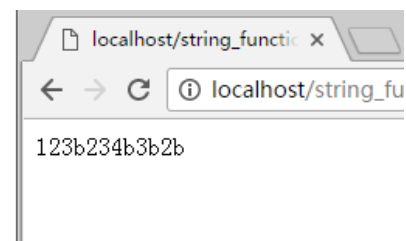
```
1 $str = '123a234a3b2a';  
2  
3 //查找位置  
4 echo strpos($str, 'a'), '<br/>';  
5 echo strrpos($str, 'a');
```



6) 替换函数: str_replace()

Str_replace(匹配目标, 替换的内容, 字符串本身): 将目标字符串中部分字符串进行替换

```
1 $str = '123a234a3b2a';  
2  
3 //字符串替换  
4 echo str_replace('a', 'b', $str);  
5
```



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

```
1 $age = 50;  
2 $name = 'TOM';  
3  
4 //格式化输出  
5 echo sprintf("你好, 今年我%d岁, 我叫%s", $age, $name);  
6  
7 //字符串替换  
8 //echo str_replace('a', 'b', $str);  
9  
10 //查找位置  
11 //echo strpos($str, 'a'). '<br/>';
```



6. A **type specifier** that says what type the argument data should be treated as. Possible types:

- **%** - a literal percent character. No argument is required.
- **b** - the argument is treated as an integer, and presented as a binary number.
- **c** - the argument is treated as an integer, and presented as the character with that ASCII value.
- **d** - the argument is treated as an integer, and presented as a (signed) decimal number. 代表十进制: %d
- **e** - the argument is treated as scientific notation (e.g. 1.2e+2). The precision specifier stands for the number of digits after the decimal point since PHP 5.2.1. In earlier versions, it was taken as number of significant digits (one less).
- **E** - like **%e** but uses uppercase letter (e.g. 1.2E+2).
- **u** - the argument is treated as an integer, and presented as an unsigned decimal number.
- **f** - the argument is treated as a float, and presented as a floating-point number (locale aware).
- **F** - the argument is treated as a float, and presented as a floating-point number (non-locale aware). Available since PHP 4.3.10 and PHP 5.0.3.
- **g** - shorter of **%e** and **%f**.
- **G** - shorter of **%E** and **%f**.
- **o** - the argument is treated as an integer, and presented as an octal number.
- **s** - the argument is treated as and presented as a string. 代表字符串%s
- **x** - the argument is treated as an integer and presented as a hexadecimal number (with lowercase letters).
- **X** - the argument is treated as an integer and presented as a hexadecimal number (with uppercase letters).

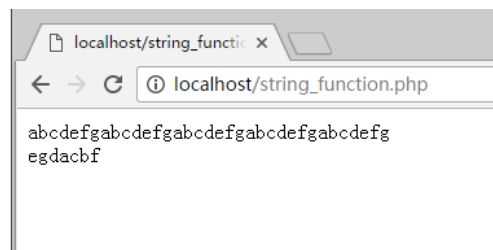
8) 其他: str_repeat(), str_shuffle()

Str_repeat(): 重复某个字符串 N 次

Str_shuffle(): 随机打乱字符串

```
//其他字符串函数
$str = 'abcdefg';

echo str_repeat($str,5), '<br/>';
echo str_shuffle($str);
```



数组详解

数组的概念

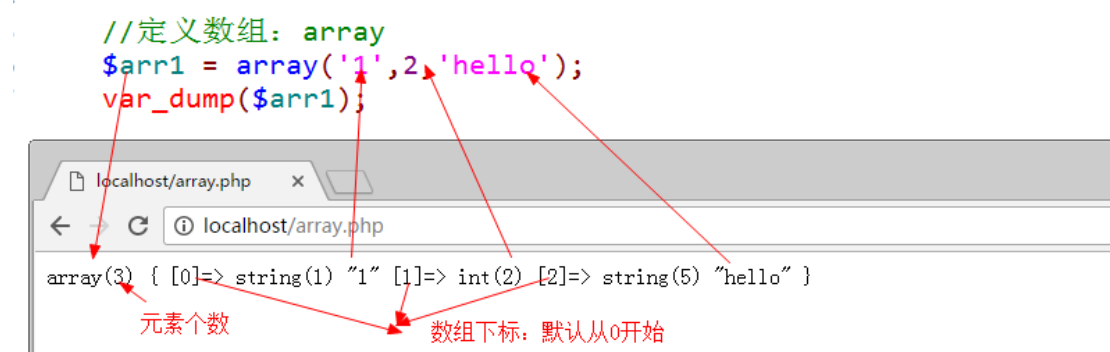
数组: array, 数据的组合, 指将一组数据(多个)存储到一个指定的容器中, 用变量指向该容器, 然后可以通过变量一次性得到该容器中的所有数据。

数组定义语法

在 PHP 中系统提供多种定义数组的方式:

1、使用 array 关键字: 最常用的

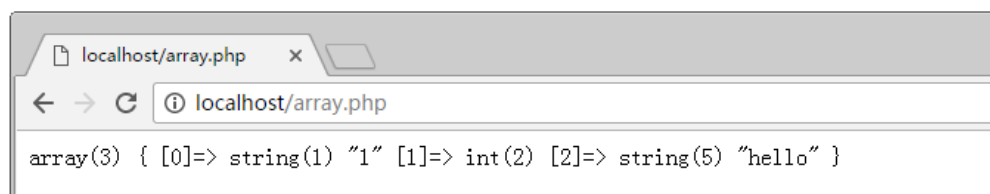
\$变量 = array(元素 1, 元素 2, 元素 3..);



2、 可以使用中括号来包裹数据:

\$变量 =[元素 1,元素 2...];

```
8
9 //定义数组: []
10 $arr2 = ['1',2,'hello'];
11 var_dump($arr2);
```

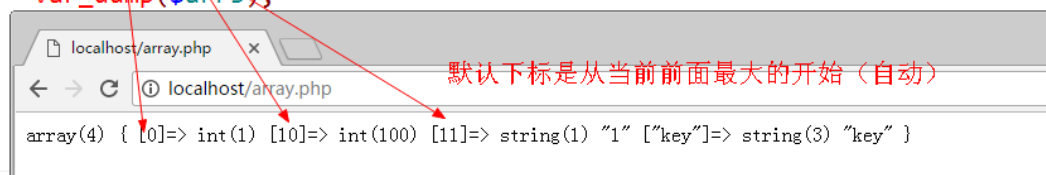


3、 隐形定义数组: 给变量增加一个中括号, 系统自动变成数组

\$变量[] = 值 1; //如果不提供下标也可以, 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, 与变量命名的规则相似

```
} //隐形数组:
1 $arr3[] = 1;
3 $arr3[10] = 100;
5 $arr3[] = '1';
7 $arr3['key'] = 'key';
8 var_dump($arr3);
```



PHP 数组特点

1) 可以整数下标或者字符串下标

如果数组下标都为整数: 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, 跟下标无关


```
3 //隐形数组:
4 $arr3[] = 1;
5 $arr3[10] = 100;
6 $arr3[] = '1';
7 $arr3['key'] = 'key';
8 $arr3[1] = 'value';
9 var_dump($arr3);
```

```
array(5) { [0]=> int(1) [10]=> int(100) [11]=> string(1) "1" ["key"]=> string(3) "key" [1]=> string(5) "value" }
```

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 和 false

空: NULL

```
//特殊下标转换
$arr4[false] = false;
$arr4[true] = true;
$arr4[NULL] = NULL;
var_dump($arr4);
```

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

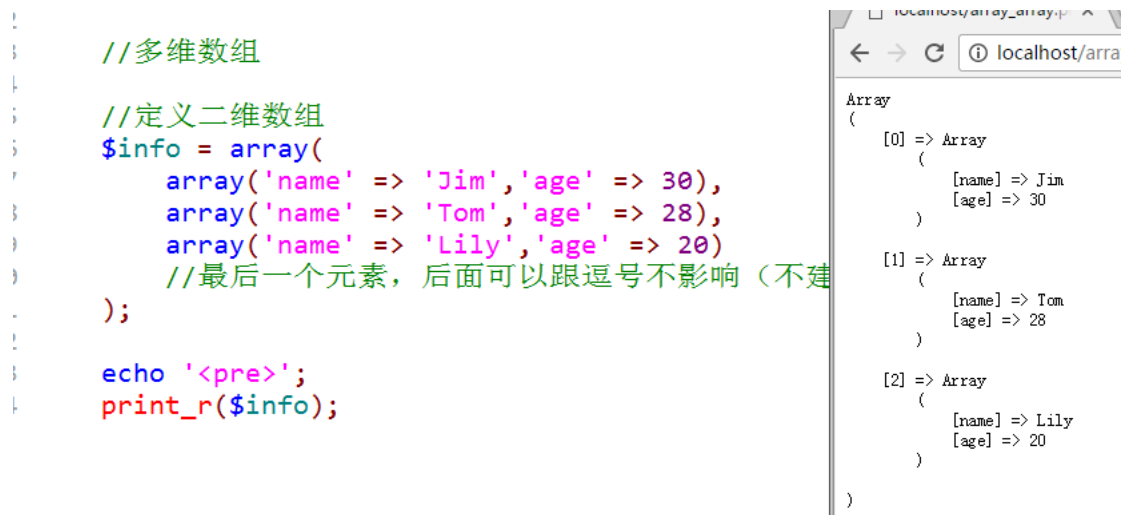
补充: PHP 中的数组是很大的数据, 所以存储位置是堆区, 为当前数组分配一块连续的内存。

多维数组

多维数组: 数组里面的元素又是数组

二维数组

二维数组: 数组中所有的元素都是一维数组



多维数组

在第二维的数组元素中可以继续是数组，在 PHP 中没有维度限制（PHP 本质并没有二维数组）

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）

数组遍历

遍历的基本含义

数组遍历：普通数组数据的访问都是通过数组元素的下标来实现访问，如果说数组中所有的数据都需要依次输出出来，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']; //Jim
```

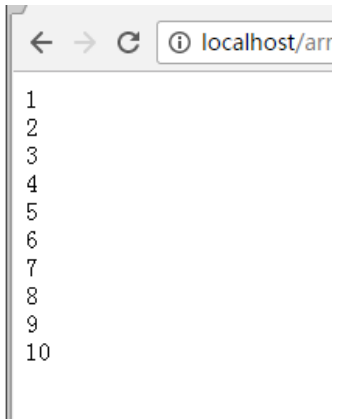
Foreach 遍历语法

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

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
  
//定义数组  
$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循环  
    foreach($arr as $k => $v){  
        //$v随意命名  
        echo 'key:', $k, ' == value:', $v, '<br/>';  
    }  
  
key:0 == value:1  
key:1 == value:2  
key:2 == value:3  
key:3 == value:4  
key:4 == value:5  
key:5 == value:6  
key:6 == value:7  
key:7 == value:8  
key:8 == value:9  
key:9 == value:10
```

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，然后二维数组元素通过下标去访问。

```
1 $arr = array(  
2     0 => array('name' => 'Tom', 'age' => 30),  
3     1 => array('name' => 'Jim', 'age' => 28)  
4 );  
5  
6 //通过foreach遍历一维元素  
7 foreach($arr as $value){  
8     //1、可以继续遍历：增加foreach遍历$value  
9     //2、可以使用下标访问  
10    echo 'name is : ', $value['name'], ' and age is : ', $value['age'], '<br/>';  
11 }
```



Foreach 遍历原理

Foreach 遍历的原理：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 $k => $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 \$k 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 的时候遇到指针取不到内容（指针指向数组最后）

数组遍历

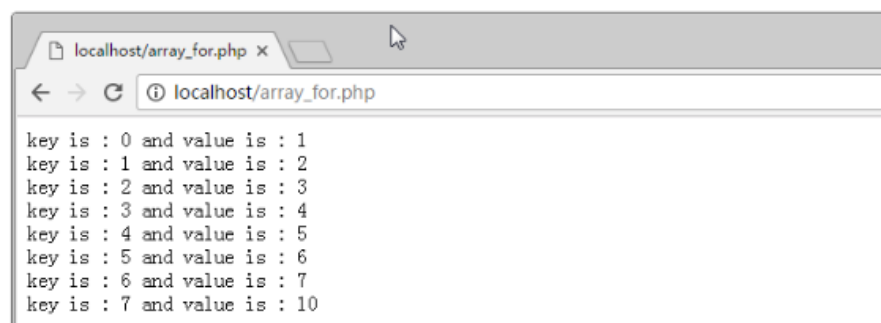
For 循环遍历数组

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

```
1 <?php  
2  
3 //for循环遍历数组  
4  
5 //数组特点：索引数组，下标规律  
6  
7 $arr = array(1,2,3,4,5,6,7,10);  
8  
9 for($i = 0,$len = count($arr); $i < $len; $i++){  
10     echo 'key is : ', $i, ' and value is : ' , $arr[$i], '<br/>';  
11 }
```



While 配合 each 和 list 遍历数组

While 是在外部定义边界条件，如果要实现可以和 for 循环一样。

Each 函数使用：each 能够从一个数组中获取当前数组指针所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以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 - 》取得元素的下标值

Value 下标 - 》取得元素的值

//while配合each和list遍历数组

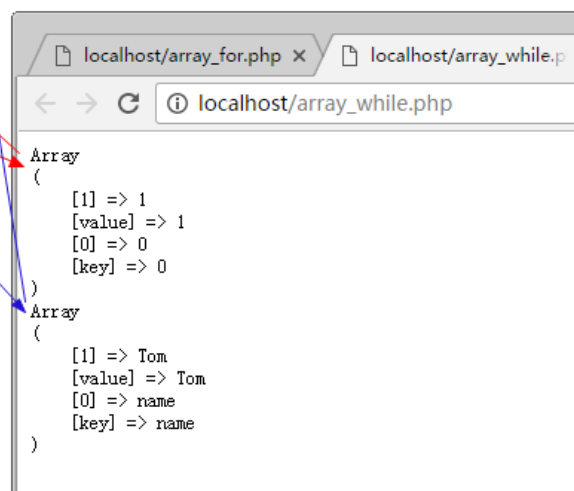
```
$arr = array(1, 'name' => 'Tom', 3, 'age' => 30);
```

```
echo '<pre>';
```

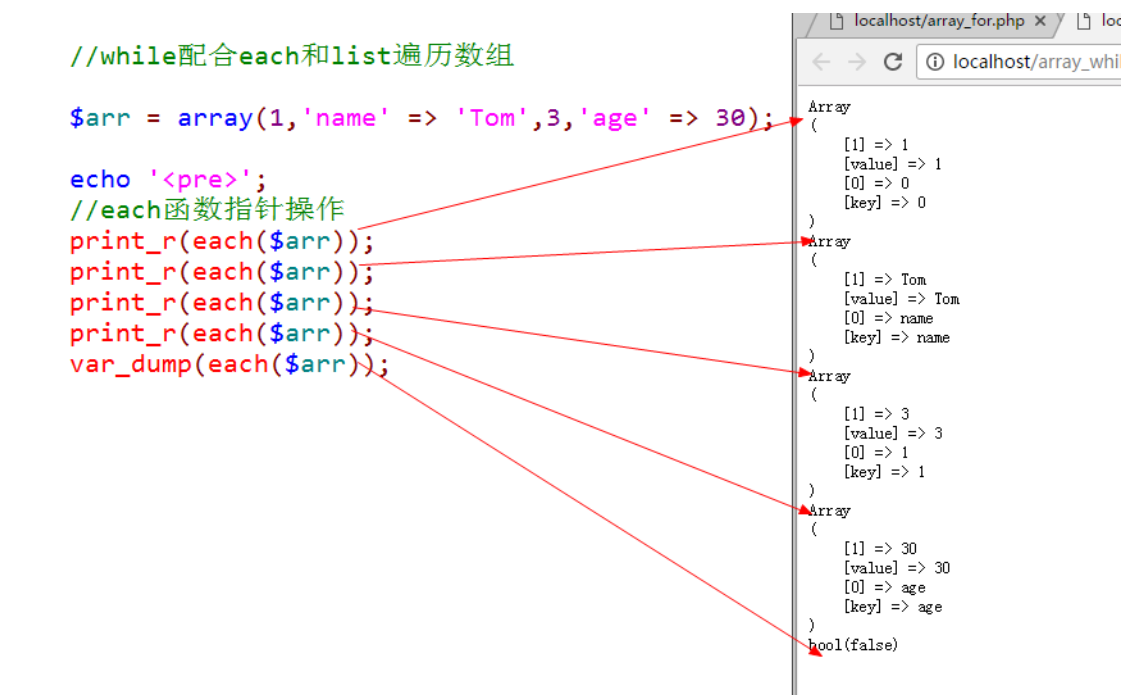
```
//each函数指针操作
```

```
print_r(each($arr));
```

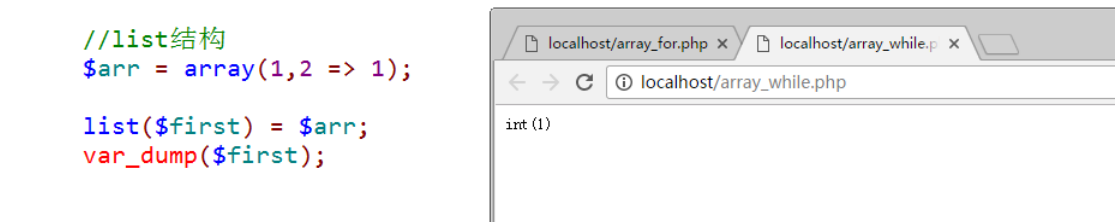
```
print_r(each($arr));
```



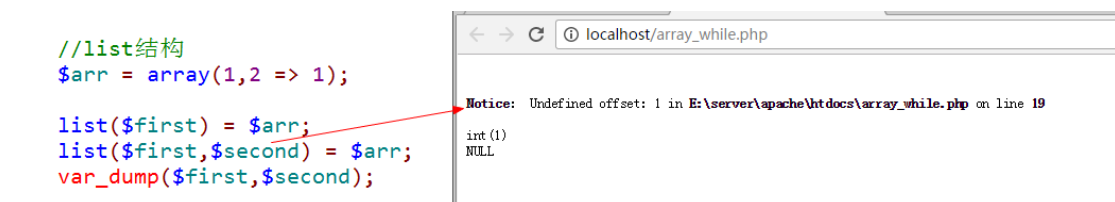
如果 each 取不到结果（数组指针移动到最后），返回 false



List 函数使用：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。
正确操作：



错误操作：变量多于数组元素，没有指定从 0 到指定变量的下标的数组元素。因为\$second 变量对应的下标是 1，但是数组中没有下标是 1



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 取不到正确的结果），整个表达式为 false

```
//while循环
$arr = array(1,'name' => 'Tom',3,'age' => 30);

while(list($key,$value) = each($arr)){
    //list搭配each
    //list($key,$value) = each($arr);

    //输出
    echo 'key is :' . $key . ' value is :' . $value . '
}
```

