

变量的作用域问题

变量的作用域，就是指：一个变量，在什么范围中可以使用的情况。

php 中，有 3 中变量作用域：

局部作用域：

就是指一个函数的内部范围。

对应这样的变量，就称为“局部变量”；

超全局作用域：

就是指所有的代码范围。

对应这样的变量，就称为“超全局变量”；

其实只有系统预定义的几个：\$_GET, \$_POST, \$_SERVER, \$_REQUEST, \$GLOBALS, \$_SESSION, \$_COOKIE, \$_FILES

全局作用域：

就是不在函数内部的范围——函数外部。

对应这样的变量，就称为“全局变量”；

通常，

- 1，全局范围不能访问局部变量；
- 2，局部范围不能访问全局变量；

3，函数内部的变量（局部变量），通常在函数调用执行结束后，就被“销毁”了。

4，但有一种局部变量，在函数调用结束后不被销毁：它叫做“静态变量”；

使用形式：

```
function 函数名 (...){  
    static $变量名 = 初始值;    //这就是静态变量!  
    . . . . .  
}
```

如果在局部作用域使用（访问）全局变量？（常见需求）

有 2 种做法：

做法 1：

使用 global 关键字来实现

做法 2：

使用\$GLOBALS 超全局变量来实现

但，如果我们对\$GLOBALS 变量的某个单元（也即下标）进行 unset，则其就会完全对应销毁该变量。

这是因为，\$GLOBALS 对全局变量的使用可以看做是全局变量的另一种语法形式而已，而不是“引用关系”，举例如下：

有关函数的系统函数：

- `function_exists()` :判断一个函数是否被定义过。其中使用的参数为“函数名”
- `func_get_arg($i)` : 获取第 *i* 个实参值
- `func_get_args()` : 获取所有实参 (结果是一个数组)
- `func_num_args()` : 获取所有实参的个数。

其他系统函数：

自己会查，并需要去查：

- 字符串函数：
 - 输出与格式化：`echo` , `print`, `printf`, `print_r`, `var_dump`.
 - 字符串去除与填充：`trim`, `ltrim`, `rtrim`, `str_pad`
 - 字符串连接与分割：`implode`, `join` , `explode`, `str_split`
 - 字符串截取：`substr`, `strchr`, `strrchr`,
 - 字符串替换：`str_replace`, `substr_replace`
 - 字符串长度与位置：`strlen`, `strpos`, `strrpos`,
 - 字符转换：`strtolower`, `strtoupper`, `lcfirst`, `ucfirst`, `ucwords`
 - 特殊字符处理：`nl2br`, `addslashes`, `htmlspecialchars`,
`htmlspecialchars_decode`,
- 时间函数：

- time, microtime, mktime, date, idate, strtotime, date_add, date_diff, date_default_timezone_set, date_default_timezone_get
- [数学函数：](#)
 - max, min, round, ceil, floor, abs, sqrt, pow, round, rand

有关函数的编程思想

递归思想——递归函数

递归函数，就是：在一个函数内部调用它自己的函数！

先考察一个最简单的函数：

```
function fl( $n ){
    echo $n;
    $n++;
    fl( $n );
}
```

fl(1);

从这个简单的函数可以看出，该函数调用是“永无止境”的（没完没了），最终会将内存消耗完毕。

显然，这不是一个正常的做法！

实用的递归函数是：能够控制这个调用的过程中，会在某个时刻（条件下）停下来！

实例演示：

求 5 的阶乘。

数学上，有这样两个有关阶乘的基本规则：

1, n 的阶乘，是 n-1 的阶乘，乘以 n 的结果。

2, 1 的阶乘是 1；

现在，假设，有一个函数，该函数“能够”计算 n 的阶乘。

```
function jiecheng( $n ){
    //.....
}
$v1 = jiecheng(8); //结果应该是 8 的阶乘
$v2 = jiecheng(5); //结果应该是 5 的阶乘
```

递归思想总结：

当面对一个“大问题”，该大问题可以经由该问题的同类问题的“小一级问题”而经过简单计算获得，而且，可以获知（已知）这类问题的“最小一级问题”的答案。则，此时就可以使用递归方法来解决该问题。则此时该函数的基本模式是：

```
function digui($n){  
if(是最小一级){  
return 已知的答案;  
}  
$jieguo = 对 digui($n-1) 进行简单运算;  
return $jieguo;  
}
```

递推（迭代）思想

递推总结：

如果要求一个“大问题”，且该问题有如下 2 个特点：

- 1，已知该问题的同类问题的最小问题的答案。
- 2，如果知道这种问题的小一级问题的答案，就可以轻松求得其“大一级”问题的答案，并且此问题的级次有一定的规律；

则此时就可以使用递推思想来解决该问题，代码模式为：

```
$qian = 已知的最小一级问题的答案;  
for($i = 最小一级的下一级; $i <= 最大一级的级次; ++$i) {  
$jieguo = 对 $qian 进行一定的计算，通常需要使用到$i;  
$qian = $jieguo;  
}  
echo “结果为: ” . $jieguo;
```

通常，如果一个问题，既能使用递归算计解决，又能使用递推算法解决，则应该使用递推算法。