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“超局部变量”；
　　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。

通常，

全剧范围不能访问局部变量；
局部范围不能访问全局变量；
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。
有一种局部变量，在函数调用结束后不被销毁：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!
    。。。。。。
}
```

```
function f3(){
    static $c = 0;//静态局部变量，它的值会保留
                  //而且，此行的赋初值，只会执行第一次
    $c++;
    $d = 0;       //此行，每次进入函数，都会执行
    $d++;
    echo "<br />c=$c, d=$d, (此函数被调用次数为：$c)";
    //到此行之后，函数结束，其中的$d变量就被销毁了！
}
f3();
f3();
f3();
```

```
function f3(){
    static $c = 0;//静态局部变量，它的值会保留
                  //而且，此行的赋初值，只会执行第一次
    $c++;
    $d = 0;       //此行，每次进入函数，都会执行
    $d++;
    echo "<br />c=$c, d=$d, (此函数被调用次数为：$c)";
    //到此行之后，函数结束，其中的$d变量就被销毁了！
}
f3();
f3();
f3();
```

```
c=1, d=1, (此函数被调用次数为：1)
c=2, d=1, (此函数被调用次数为：2)
c=3, d=1, (此函数被调用次数为：3)
```

可见，静态变量可以用于统计（计算）一个函数被调用了多少次。
当然，也而已用于其它需要进行数据保留的函数调用过程。

如何在局部作用域使用（访问）全局变量？（常见需求）

第一种：使用 **global** 关键字

有 2 种做法：

做法 1：

```php
echo "<hr />";
$v4 = 4;
function f4(){
    //在函数中，使用global来声明一个要使用的全局变量的同名局部变量。
    global $v4; //这里，$v4是局部变量，只是跟全局的v4同名
                //实际情况是：此时外部v4变量跟内部的v4变量，共同
                //指向一个数据区—引用关系！
    echo "<br />在局部访问全局变量v4 = $v4";
    $v4 = 44;    //修改其值；
}
f4();
echo "<br />在全局再次访问v4 = $v4";

?>
```

全局变量$v4 → 数据：4 → 44

局部变量$v4

如果这里unset($v4); 不影响外部$v4的使用

网页标题 × PHP核心 www.php39.c

在局部访问全局变量v4 = 4
在全局再次访问v4 = 44

```php
echo "<hr />";
$v4 = 4;
function f4(){
    //在函数中，使用global来声明一个要使用的全局变量的同名局部变量。
    global $v4; //这里，$v4是局部变量，只是跟全局的v4同名
                //实际情况是：此时外部v4变量跟内部的v4变量，共同
                //指向一个数据区—引用关系！
    echo "<br />在局部访问全局变量v4 = $v4";
    $v4 = 44;    //修改其值；
    unset($v4);
}
f4();
echo "<br />在全局再次访问v4 = $v4";
```

在局部访问全局变量v4 = 4
在全局再次访问v4 = 44

第二种：使用**$GLOBALS 超全局变量**

```php
echo "<hr />";
$v5 = 5;
function f5(){
    echo "<br />在局部访问全局变量v5 = " . $GLOBALS['v5'];
    $GLOBALS['v5'] = 44;    //修改其值；
}
f5();
echo "<br />在全局再次访问v5 = $v5";

?>
</body>
</html>
```

网页标题 × PHP核心知识 × www.php39.com/day6/

在局部访问全局变量v5 = 5
在全局再次访问v5 = 44

```php
echo "<hr />";
$v5 = 5;
function f5(){
    echo "<br />在局部访问全局变量v5 = " . $GLOBALS['v5'];
    $GLOBALS['v5'] = 55;    //修改其值;
}
f5();
echo "<br />在全局再次访问v5 = " . $v5;
echo "<br />在全局再次访问v5 = " . $GLOBALS['v5'];

?>
</body>
</html>
```


```php
$v5 = 5;    //全局变量
function f5(){
    //$GLOBALS可以认为是全局变量的另一种使用形式!
    echo "<br />在局部访问全局变量v5 = " . $GLOBALS['v5'];
    $GLOBALS['v5'] = 55;    //修改其值;
}
```

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”

```php
$v6 = 6;    //全局变量
function f6(){
    //$GLOBALS可以认为是全局变量的另一种使用形式!
    echo "<br />在局部访问全局变量v6 = " . $GLOBALS['v6'];
    $GLOBALS['v6'] = 66;    //修改其值;
    unset($GLOBALS['v6']);
}
f6();
echo "<br />在全局再次访问v6 = " . $v6;
?>
</body>
</html>
```

此时就是实实在在地销毁了该全局变量

有关函数的系统函数

```php
<body>
<?php
if( function_exists("f1") == false ){
    function  f1(){
        echo "<br />这个函数我自己定义了!";
    }
}
f1();   //这里就可以放心使用该函数了!
?>
</body>
</html>
```

查询函数手册

有关函数的编程思想

递归思想------递归函数：在函数内部调用它自己的函数！

一个最简单的递归函数：function f1($n){

Echo $n;

$n++;

F1($n);//函数调用自己，消耗内存，该函数的调用是永无止境的，最终将内存消耗完毕。

}//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
```

```php
<?php
//现在，假设，有一个函数，该函数“能够”计算n的阶乘。
function  jiecheng( $n ){
    if( $n == 1){
        return 1;
    }
    $jieguo = $n * jiecheng($n-1);
    return $jieguo;
}
$v2= jiecheng(5);    //结果应该是5的阶乘
/*
演示调用过程：
$v1 = jiecheng(5)相当于：
$v1 = 5 * jiecheng(4)==>>
$v1 = 5 * (4 * jiecheng(3) ) ==>>
$v1 = 5 * (4 * (3 * jiecheng(2) ) ) ==>>
$v1 = 5 * (4 * (3 * (2 * jiecheng(1) ) ) ) ==>>
$v1 = 5 * (4 * (3 * (2 * 1 ) ) ) ==>>
$v1 = 5 * (4 * (3 * 2 ) ) ==>>
$v1 = 5 * (4 * 6 ) ==>>
$v1 = 5 * 24 ==>>
$v1 = 120
*/
?>
```

```php
<?php
//现在，假设，有一个函数，该函数“能够”计算n的阶乘。
function  jiecheng( $n ){
    echo "<br />开始：有人要求{$n}的阶乘";
    if( $n == 1){
        echo "<br />结束：终于求到了{$n}的阶乘：：1";
        return 1;
    }
    $jieguo = $n * jiecheng($n-1);
        echo "<br />结束：终于求到了{$n}的阶乘：：$jieg
    return $jieguo;
}
$v2= jiecheng(5);    //结果应该是5的阶乘
/*
演示调用过程：
$v1 = jiecheng(5)相当于：
$v1 = 5 * jiecheng(4)==>>
$v1 = 5 * (4 * jiecheng(3) ) ==>>
$v1 = 5 * (4 * (3 * jiecheng(2) ) ) ==>>
$v1 = 5 * (4 * (3 * (2 * jiecheng(1) ) ) ) ==>>
$v1 = 5 * (4 * (3 * (2 * 1 ) ) ) ==>>
$v1 = 5 * (4 * (3 * 2 ) ) ==>>
$v1 = 5 * (4 * 6 ) ==>>
$v1 = 5 * 24 ==>>
$v1 = 120
*/
echo "<br />v2 = $v2";
?>
</body>
</html>
```

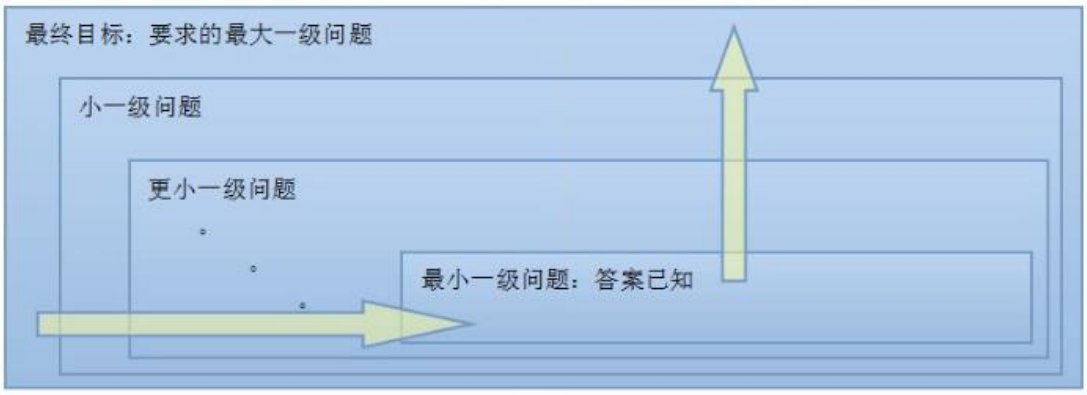
递归思想的总结

当面对一个“大问题”，该大问题可以经由该问题的“小一级问题”而经过简单的计算获得，而且可以获知这类问题的“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;
}
```

递归思想图示



递推（迭代）思想

```php
<?php
//演示递推思想：
//目标：要求5的阶乘：
$jiecheng = 1;  //表示1的阶乘（已知）；
for($i = 2; $i <= 5; ++$i){
    $jiecheng = $jiecheng * $i; //i的阶乘；
}
?>
```

```php
<?php
//演示递推思想：
//目标：要求5的阶乘：
$qian = 1;  //表示1的阶乘（已知）；
for($i = 2; $i <= 5; ++$i){
    //此循环会从2的阶乘开始，一次次求得
    //“更大”一个数的阶乘，直到5的阶乘
    $jiecheng = $jiecheng * $i; //i的阶乘；
}
?>
```

```php
<?php
//演示递推思想：
//目标：要求5的阶乘：
$qian = 1;  //表示“前一个已知的答案”：这里是第一个，就是1的阶乘
for($i = 2; $i <= 5; ++$i){
    //此循环会从2的阶乘开始，一次次求得
    //“更大”一个数的阶乘，直到5的阶乘
    $jieguo = $qian * $i;//要求的结果是由“前一个结果”经过简单乘法运算而得到
    $qian = $jieguo;    //将当前求得的“结果”，又当成“前一个”，以供下一次使用！
}
?>
```

```php
<?php
//演示递推思想：
//目标：要求5的阶乘：
$qian = 1;  //表示“前一个已知的答案”：这里是第一个，就是1的阶乘
for($i = 2; $i <= 5; ++$i){//意图求得从2开始到5的“每一个数阶乘”
    //此循环会从2的阶乘开始，一次次求得
    //“更大”一个数的阶乘，直到5的阶乘
    $jieguo = $qian * $i;//要求的结果是由“前一个结果”经过简单乘法运算而得到
    echo "<br />{$i}的阶乘是$jieguo";
    $qian = $jieguo;    //将当前求得的“结果”，又当成“前一个”，以供下一次使用！
}
echo "<br />结果为：" . $jieguo;
?>
</body>
</html>
```

网页标题

2的阶乘是2
3的阶乘是6
4的阶乘是24
5的阶乘是120
结果为：120

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；

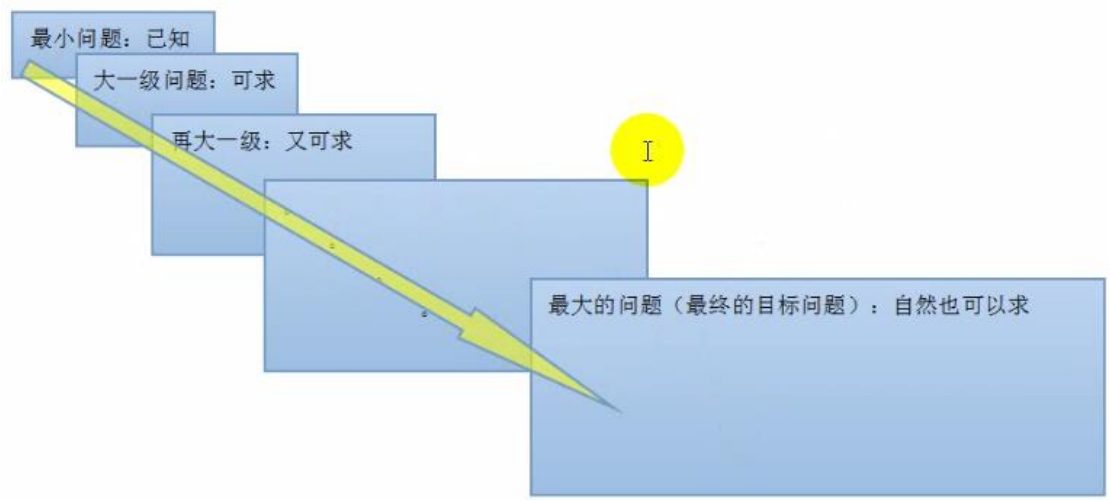
则此时就可以使用递推思想来解决该问题，代码模式为：

```
$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;
```

```php
/*
下面用递推思想来完成刚才的数列题：
以下数列：1，  1，  2，  3，  5，  8，  13，  .......
求第20项：
*/
$qian1 = 1;
$qian2 = 1;
for($i = 3; $i <= 20; ++$i){
    $jieguo = $qian1 + $qian2;  //第3项
     $qian1 = $qian2;
     $qian2 = $jieguo;
}


?>
```

递推思想图示



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。

效率高！