

位运算符的应用—管理一组事物的开关状态

开关状态定义：只有 2 中结果，对应就是布尔值类型的值。True false

管理目标：使用一个变量就可以表达若干个灯泡的“当前状态”，具体有 3 个任务：

1.通过该变量，可以获得任何一个数据（灯泡）当前状态：

```
7  define("D1", 1);      //对应二进制的值为：00000001
8  define("D2", 2);      //对应二进制的值为：00000010
9  define("D3", 4);      //对应二进制的值为：00000100
10 define("D4", 8);      //对应二进制的值为：00001000
11 define("D5", 16);     //对应二进制的值为：00010000
12 //定义个变量，该变量代表5个灯泡的任意值组合
13 $state=10; //对应二进制值：00001010
14 //
15 //任务1：指定任意一个灯泡的当前状态，并输出
16 if (($state & D1) > 0) {
17     echo "<br/>灯亮";
18 }else{
19     echo "<br/>灯灭";
20 }
```

2.所有灯都整体显示状态：

```
34 function ShowAll(){
35     for ($i=1; $i <=4 ; $i++) {
36         $s="D".$i;
37         //GLOBALS['state'] 调用外部函数state
38         if (($GLOBALS['state']&constant($s))>0) {
39             echo "<br/>灯{$s}亮";
40         } else {
41             echo "<br/>灯{$s}灭";
42         }
43     }
44 }
45 ShowAll()
```

3.通过该变量，可以将一个一个数据的状态“开启”，即：开启任意一个指定灯泡：

```
47 // 开启指定灯泡
48 $state=$state | D1;
49 echo "<br/>开启灯1,后的状态";
50 ShowAll();
```

4.通过该变量，可以将一个一个数据的状态“关闭”，即：关闭任意一个指定灯泡：

```
51 // 关闭指定灯泡
52 $state=$state & (~D1);
53 echo "<br/>关闭灯1,后的状态";
54 ShowAll();
```

数组运算符

+: 数组联合：即“数组串联”

将右边的数组项并到左边数组后边，得到数组，如有重复键，则结果以左边为准

```
$arr1 = array(5=>10, 8=>20, 10=>30);
$arr2 = array(3=>33, 2=>22);
$r1 = $arr1 + $arr2; //结果为: array(5=>10, 8=>20, 10=>30, 3=>33, 2=>22)
```

`==`: 如果两个数组具有相同的键名和键值（可以顺序不同，或类型不同），则返回 `true`
`!=`:
`===`: 如果两个数组具有相同的键名和键值且顺序和类型都一样，则返回 `true`
`!==`:

错误控制运算符

通常就用一个地方: `$link = @mysql_connect("localhost","root","123");`

作用: 如果该链接数据库的语句失败, 则屏蔽该失败的错误提示!

运算符优先级

1. 要有运算符优先级意识
2. 括号最优先, 赋值最落后
3. 先乘除后加减
4. 大致: 单项运算符 > 算术运算符 > 比较运算符 > 逻辑运算符(出了“非”运算符)

流程控制

if 条件控制: 如果表达式 `expr` 的值为 `true`, 则执行 `statement` 语句, 否则就跳过该语句继续往下执行。

```
if (expr) {  
    statement;  
}
```

if ... else 条件控制: 如果表达式 `expr` 的值为 `true`, 则执行 `statement1` 语句, 否则执行 `statement2`。

```
if (expr) {  
    statement1;  
} else {  
    statement2;  
}
```

Switch ... case 分支控制语句: Switch 语句根据 `variable` 的值, 依次与 `case` 中的 `value` 值相比较, 如果不相等, 继续查找下一个 `case`; 如果相等, 就执行对应语句, 直到 `switch` 语句结束或遇到 `break` 为止, 一般 `switch` 语句结尾都有一个默认的 `default`, 如果在前边 `case` 没找到相符合的条件, 则输出默认语句, 这跟 `else` 语句类似。

```
switch (variable) {  
    case 'value1':  
        statement1;  
        break;  
    case 'value2':  
        ...  
    default:  
        default statement n;  
        break;  
}
```

for 循环: 嵌套循环, 九九乘法表

```
for ($i=1; $i <=9 ; $i++) {  
    for ($k=1; $k <= $i ; $k++) {  
        echo "$i x $k =" . ($i*$k) . "&nbsp;&nbsp;&nbsp;";  
    }  
    echo "<br/>";  
}
```

循环中，有两种中断语句可以使用：

Break：用于完全终止某个循环，让执行流程进入到循环语句后的语句，

Continue：用于停止当前正在进行的当次循环，而进入到循环的下一一次（开始）循环中，


循环的“层”，指的是从当前中断语句（**break** 或 **continue**）算起，往代码的“外部”数循环的个数，就是层数。比如：

```
for(...){           //循环 1
    for(...){       //循环 2
        for(...){   //循环 3
            break 2; //此时会中断循环 2：其实指中断“2层”
                    //对此 break 语句，循环 3 是其“第一层”，循环 2 是其第 2 层，循环 1 是其第 3 层
        }
        continue 2; //此时会中断循环 12：其实指中断“2层”
                    //对此 continue 语句，循环 2 是其“第 1 层”，循环 1 是其第 2 层，
    }
}
```

do while 循环：

for 循环语句形式：

```
for (【循环变量初始化】； 【循环变量的条件判断】； 【循环变量的改变】){
    //循环体语句。。。
}
```



while 循环语句形式：

```
【循环变量初始化】
while(【循环变量的条件判断】){
    //循环体语句。。。
    【循环变量的改变】
}
```

do while 循环语句形式：

```
【循环变量初始化】
do {
    //循环体语句。。。
    【循环变量的改变】
}while(【循环变量的条件判断】);
```

说明：

- 1.do ... while 会先进入循环执行一次（不判断条件）。
- 2.然后判断循环条件是否满足，如果循环，就进入 do 的开始位置，进行第二次循环。
- 3.如果条件不满足，循环就结束。

Sleep(\$n)：

让程序停止运行指定的秒数，然后继续运行。

```
echo "<br />A";
echo "<br />当前时间: " . Date("Y-m-d H:i:s");
sleep(3); //停止3秒，然后才允许后续程序
echo "<br />当前时间: " . Date("Y-m-d H:i:s");
echo "<br />B";
die("<br />终止!");
echo "<br />X1";
echo "<br />X2";
```

文件加载:

基本语法: include, require, include_once, require_once

文件加载:

绝对路径

```
echo "<p>使用相对路径载入";  
include './page1.php';  
  
echo "<p>使用绝对路径载入(方法1)";  
include __DIR__ . '\page1.php';  
  
echo "<p>使用绝对路径载入(方法2)";  
$root = $_SERVER['DOCUMENT_ROOT']; //获得当前站点的跟目录  
include $root . "\day5" . '\page1.php';
```

使用相对路径载入
这是被载入页面page1.php

使用绝对路径载入(方法1)
这是被载入页面page1.php

使用绝对路径载入(方法2)
这是被载入页面page1.php

文件载入和执行过程:

第1步: 从 include 语句退出 php 脚本模式 (进入 html 代码模式)

第2步: 载入 include 语句所设定的文件中的代码, 并执行之 (如同在当前文件中一样)

第3步: 退出 html 模式重新进入 php 脚本模式, 继续之后的代码。

```
<body>  
主文件开始位置:  
<?php  
    echo "<br />主文件中的位置A";  
  
    include "./page2.php"; //要载入的文件  
  
    echo "<br />主文件中的位置B";  
?>  
<br />主文件结束位置:
```

```
H:\itcast\class\bj-php-39\day5\page2.php  
1 <br />被载入文件位置1  
2 <?php  
3 echo "<br />被载入文件位置2";  
4 ?>  
5 <br />被载入文件位置3
```

相当于:

```
<body>  
主文件开始位置:  
<?php  
    echo "<br />主文件中的位置A";  
    此处, 本来是这样: include './page2.php';  
1 ?>  
    <br />被载入文件位置1  
    <?php  
    echo "<br />被载入文件位置2";  
    ?>  
    <br />被载入文件位置3  
3 <?php  
    echo "<br />主文件中的位置B";  
?>  
<br />主文件结束位置:
```

```
H:\itcast\class\bj-php-39\day5\page2.php  
1 <br />被载入文件位置1  
2 <?php  
3 echo "<br />被载入文件位置2";  
4 ?>  
5 <br />被载入文件位置3
```

4 个载入语句的区别:

require, 和 include 的区别:

引用失败 (出错) 时, include 警告并继续, require 终止。

通常, require 用于在程序中, 后续代码依赖于载入的文件的时候。

include,和 include_once 的区别:

Include 载入的文件不判断是否重复, 只要有 include 语句, 就会载入一次(可能导致重复载入)

include_once 载入的文件有内部判断机制是否“前面代码”已经载入, 加载一次不重复加载

Require 和 require_once 的区别:

同 include 和 include_once 的区别。