

运算符

算术运算符

符号: + - * / %

说明: 1.都是针对数字进行的运算

2.如果他们两边有不是数字的运算数据, 就会自动转换为数字

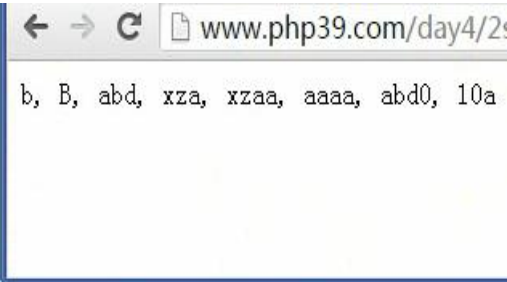
3.取余运算%, 值针对整数进行运算, 如果不足, 会自动截取整数, 11.8%3.3==11%3

自增自减运算符

- 常规: 对数字进行自加1或自减1。
- 字符串: 只能自增, 且自增的效果就是 "下一个字符"
- 布尔值递增递减无效
- null递增递减无效, 递增结果为1

字符串自增例子:

```
$s1 = "a";
$s2 = "A";
$s3 = "abc";
$s4 = "xyz";
$s5 = "xyzz";
$s6 = "zzz";
$s7 = "abc9";
$s8 = "9z";
$s1++;$s2++;$s3++;$s4++;$s5++;$s6++;$s7++;$s8++;
echo "$s1, $s2, $s3, $s4, $s5, $s6, $s7, $s8";
```

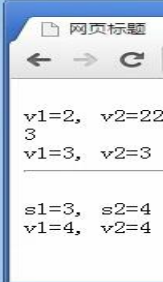


前自增和后自增的区别 (自减类似)

```
$v1 = 1;
$v2 = 1;
$v1++; //此行后, v1为2
++$v2; //此行后, v2为2
echo "<br />v1=$v1, v2=$v2";
//说明: 此时前自增后自增效果一样!

echo $v1++; //输出2, 此行后, v1为3
echo "<br />";
echo ++$v2; //输出3, 此行后, v2为3
echo "<br />v1=$v1, v2=$v2";

echo "<br />";
$s1 = $v1++; //s1为3, 此行后, v1为4
$s2 = ++$v2; //s2为4, 此行后, v2为4
echo "<br />s1=$s1, s2=$s2";
echo "<br />v1=$v1, v2=$v2";
//可见, 在有加加运算的其他语句中,
//前加加和后加加会有区别:
//影响其他语句的执行结果:
//前加加是先对自加变量加1, 然后做其他运算
//后加加是先做其他运算, 然后对自加变量加1
```



在循环中, 推荐使用前加加, (耗时短, 效率高)

```
$t1 = microtime(true); //获得当前时间, 精确到万分之一秒
for($i = 1; $i < 10000000; ++$i){
}
$t2 = microtime(true); //获得当前时间, 精确到万分之一秒
for($i = 1; $i < 10000000; $i++){
}
$t3 = microtime(true); //获得当前时间, 精确到万分之一秒
echo "<p>前加加耗时: " . ($t2-$t1);
echo "<p>后加加耗时: " . ($t3-$t2);
```

前加加耗时: 0.60762214660645

后加加耗时: 0.98578596115112

比较运算符

- 符号: > >= < <= == != === !==
- 一般比较: 是针对数字进行大小比较
- ==和===比较: 前者模糊相等的比较, 后者叫做精准相等的比较
- 不要对浮点数直接进行比较

常见不同类型数据之间的比较规律:

- 如果比较的数据有布尔值, 就转为布尔值比较, 布尔值比较只有一个规则: true>>false
- 否则。如果有数字值, 就转为数字值比较----常规比较
- 否则, 如果两边都是“纯数字字符串”, 转为数字比较
- 否则就按字符串比较, 字符串比较规则:
 - 对两边的字符串, 一个一个从前往后取出字符并进行比较, 谁“先大”, 结果就是它大。

```
"abc" > true    //? false
"abc" > false   //true
"0" > false     //false
3 > "12";      //false
3 > "12abc";    //false
"3" > "12"     //false
"abc" > "c";    //false, 后者大
"abc" > "ab123cde"; //true 因为这里"c"大于'
"3abc">"12abc"; //true, 因为"3"大于 "1"
1 > "a";       //? true
"1" > "a"      //? false
```

逻辑运算符

逻辑运算符针对“布尔值”进行运算, 如果不是布尔值就转为布尔值进行运算, 布尔值: true、false

基本运算规则（真值表）

逻辑与运算

```
true  &&  true    ===>> true
true  &&  false   ===>>false
false &&  true    ===>> false
false &&  false   ===>>false
```

总结: 只有 2 个都是 true 时, 结果就是 true。只要有一个 false, 结果就是 false

逻辑或运算

```
true  ||  true    ===>> true
true  ||  false   ===>>true
false ||  true    ===>> true
false ||  false   ===>>false
```

总结: 只有 1 个都是 true 时, 结果就是 true。只要有 2 个 false, 结果就是 false

逻辑非运算

```
!true===>>false
!false===>>true
```

逻辑运算符的“短路现象”：

```
14 //此函数只是为了说明要对2个数据(x,y)进行
15 //复杂的计算，然后返回计算结果
16 function f1($x, $y){
17     $m1 = $x*2;
18     $m2 = $y*3;
19     return $m1 + $m2;
20 }
21 $n1 = 3;
22 $n2 = 4;
23 //if判断语句写法1:
24 if( $n1 > $n2 && f1($n1, $n2) > 20 ){
25     //这里完成某种任务1
26 }
27 else{
28     //这里完成另一些任务2
29 }
30 //if判断语句写法2:
31 if( f1($n1, $n2) > 20 && $n1 > $n2 ){
32     //这里完成某种任务1
33 }
34 else{
35     //这里完成另一些任务2
36 }
37 //写法1和写法2，最终计算结果是一样的！
38 //但写法1会具有优势：它有时候可能不需要进行“复杂”计算
39 //就可以得到判断结果，这就是“短路”现象
40 //而写法2却总是先去进行“复杂”计算，显然属于消耗资源行为
```

结果：如果一个语句中，通过“与”运算需要进行多次复杂判断，我们将简单的判断放在前边，利用“短路现象”提高效率。

字符串运算符

- 1, 符号只有一个：. 也衍生出另一个：.=
 - 2, 含义：就是将这个符号两边的字符串连接起来；
 - 3, 如果两边不是字符串，就会自动转换为字符串，然后连接起来。
- “ab” . 3 ===>“ab3” “12” . 3 ===>“123”

赋值运算符

基本赋值运算符：=

形式：\$ 变量名 = 值

理解：将右边的值，赋值给左边的变量

衍生运算符：

+= 加等：形式：\$ 变量名 += 值

理解：\$ 变量名 = \$变量名 + 值

-= 减等：形式：\$ 变量名 -= 值

理解：\$ 变量名 = \$变量名 - 值

*= /= %= .= 其都可以认为是上述形式的一中简化版。

条件运算符

数据值 1 ? 数据值 2 : 数据值 3

含义：对数据 1 判断，如果为真，运算结果就是数据值 2，否则就是数据值 3

If (数据值 1) {

\$变量名 = 数据值 2;

}else{

\$变量名 = 数据值 3;

}

位运算符

1.位：指二进制数字的每一个“位”

2.位运算仅仅针对整数进行运算；

3.位运算符有如下几个：

&：按位与， |：按位或， ~：按位非， ^：按位异或

整数的按位与运算（&）

形式：

`$n1 & $n2;` //n1, n2 是 2 个任意整数；

含义：

将该 2 个整数的二进制数字形式（注意，都是 32 位）的每一个对应位上的数字进行基本按位与运算之后的结果！

注意：他们运算的结果，其实仍然是一个普通的数字（10 进制）。

示例图示（只用 8 个位来演示）：

`$r1 = 10 & 20;`

10 的 2 进制	0	0	0	0	1	0	1	0
20 的 2 进制	0	0	0	1	0	1	0	0
& 运算结果：	0	0	0	0	0	0	0	0

代码验证：

```
$v1 = 10 & 20;
echo "<br />v1 = $v1";
```

v1 = 0

整数的按位或运算（|）

同理。按位或运算代码验证：

```
$v1 = 12 | 24;
echo "<br />v1 = $v1";
```

v1 = 28

整数的按位左移运算（<<）

形式：`$n1 << $m` 含义：将十进制数字 n1 的二进制数字形式(32 位)的每一个位上的数字左移 m 位，并将右边空出来的补 0，左边溢出来的删除。

示例（8 位演示）

`$r1 = 10 << 2;`

10 的 2 进制	0	0	0	0	1	0	1	0
左移 2 位后	0	0	1	0	1	0	0	0
则结果为：			2^5	0	2^3			

结果为： $2^5+2^3=32+8=40$

代码验证：

```
$v1 = 10 << 2;
echo "<br />v1 = $v1";
```

v1 = 40