

# 运算符

## 算术运算符

### 基础：

符号有：+   -   \*   /   %

说明：

- 1、他们都是针对数字进行的运算；
- 2、如果他们的两边有不是数字的数据，就会（自动）转换为数字；
- 3、其中取余运算（取模运算）%，它只针对“整数”进行运算，如果不是，会自动截取为整数。

11.3 % 3 相当于 11 % 3;

11.8 % 3.8 相当于 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”，其只能针对字母或数字进行自加：

布尔值递增递减无效 null 递减无效，递增结果为 1

### 前自增和后自增的区别（自减类似）

说明：

- 1、独立的加加运算中，前自增后自增效果一样
- 2、在有加加运算的其他语句中，前加加和后加加会有区别：  
前加加是先对自加变量加 1，然后再做其他运算  
后加加是先做其他运算，再对自加变量加 1

通常，我们在循环中，推荐使用前加加，比如：

```
for($i = 1; $i < 10000; ++$i){ ..... }
```

## 比较运算符

符号: > >= < <= == != === !==

一般比较: 是针对数字进行的大小比较

==和===比较: 前者通常叫做模糊相等的比较, 后者叫做精确相等的比较 (只有数据的类型和数据的值/内容, 都相等, 才是全等的)。必须能够找到手册的“类型比较表”: 附录》php 类型比较表

不要对浮点数直接进行大小比较

## 常见不同类型 (标量类型) 数据之间的比较规律:

如果比较的数据中, 有布尔值, 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: 这就是常规比较。

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。字符串比较的规则是:

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。

## 逻辑运算符

逻辑运算符都是针对“布尔值”进行的运算。

如果不是布尔值, 就会转换为布尔值进行;

布尔值只有 2 个: `true`, `false`

## 基本运算规则 (真值表):

逻辑与规则:

`true && true ==> true`

`true && false ==> false`

`false && true ==> false`

`false && false ==> false`

总结: 只有 2 个都是 `true`, 结果才是 `true`

只要有一个是 `false`, 结果就是 `false`

逻辑或规则:

`true || true ==> true`

`true || false ==> true`

`false || true ==> true`

`false || false ==> false`

总结: 只有 2 个都是 `false`, 结果才是 `false`

只要有一个是 `true`, 结果就是 `true`

逻辑非规则:

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

## 逻辑运算符的“短路现象”：

逻辑与短路：

结果：如果一个语句中，通过与运算需要进行多项判断，而且不同的判断具有明显不同的“复杂程度”，则我们应该将简单的判断放在前面，这时候我们就可以利用短路现象以达到提高效率的目的。

逻辑或短路：

结果：如果一个语句中，通过或运算需要进行多项判断，而且不同的判断具有明显不同的“复杂程度”，则我们应该将简单的判断放在前面，这时候我们就可以利用短路现象以达到提高效率的目的。

## 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"
12 . 3        ==> "123"
```

## 赋值运算符：

一个基本赋值运算符：`=`

形式：`$变量名 = 值；`

理解：将右边的值（不管做了多少运算），赋值给左边的变量。

若干个衍生的赋值运算符：

`+=` 加等：形式：`$变量名 += 值；`

理解：相当于：`$变量名 = $变量名 + 值；`

`-=` 加等：形式：`$变量名 -= 值；`

理解：相当于：`$变量名 = $变量名 - 值；`

`*=`   `/=`   `%=`   `.=`   其都可以认为是上述形式的一种简化版。

## 条件（三目，三元）运算符

只有一个，形式如下：

数据值 1   ?   数据值 2   :   数据值 3

含义：

对数据值 1 进行判断，如果为“真”，则该运算符的运算结果就是数据值 2，否则就是数据值 3；

它是这样一个流程控制（逻辑判断）语句的简写形式：

```
if ( 数据值 1 ) {  
    $变量名 = 数据值 2 ;  
}  
else {  
    $变量名 = 数据值 3 ;  
}
```

注意：如果数据值 1 不是布尔值，也会转换为布尔值；

```
$score = 66;        //分数
```

```
$valuation = $score >= 60 ? “及格” : “不及格”; //结果为“及格”
```

```
$score = 56;        //分数
```

```
$valuation = $score >= 60 ? “及格” : “不及格”; //结果为“不及格”
```

```
$score = 56;        //分数
```

`$valuation = $score ? “及格” : “不及格”;    //结果为“及格”，这里可能就偏离的本意！！`

## 位运算符

### 基础规定

1，位是什么？就是 2 进制数字的每一个“位”，一个整数数字，有（由）32 个位构成！

2，位运算符是仅仅针对整数进行的运算符；

3，位运算符有如下几个

&: 按位与；

|: 按位或；

~: 按位非；按位取反；

^: 按位异或；

4，位运算符的基本语法规则：

按位与基本规则：

1 & 1 ==> 1  
1 & 0 ==> 0  
0 & 1 ==> 0  
0 & 0 ==> 0

按位或基本规则：

1 | 1 ==> 1  
1 | 0 ==> 1  
0 | 1 ==> 1  
0 | 0 ==> 0

按位非基本规则：

~1 ==> 0  
~0 ==> 1

按位异或基本规则：

1 ^ 1 ==> 0  
1 ^ 0 ==> 1  
0 ^ 1 ==> 1  
0 ^ 0 ==> 0

可见，按位异或的规则是：相同为 0，不同为 1

## 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）。

## 整数的按位或运算：

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或运算之后的结果！

注意：他们运算的结果，其实仍然是一个普通的数字（10 进制）。

## 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（也是 32 位的）的每一个位上的数字都一次性往左边移动 **m** 位，

并将右边空出来的位置补 0，左边冒出去的不管，这样操作之后得到的结果。