运算符

算数运算符

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;

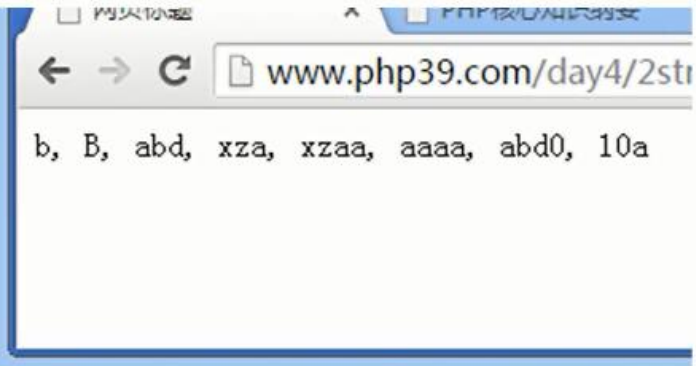
11.8　%　3.8　相当于　11%3;

自增自减运算符：

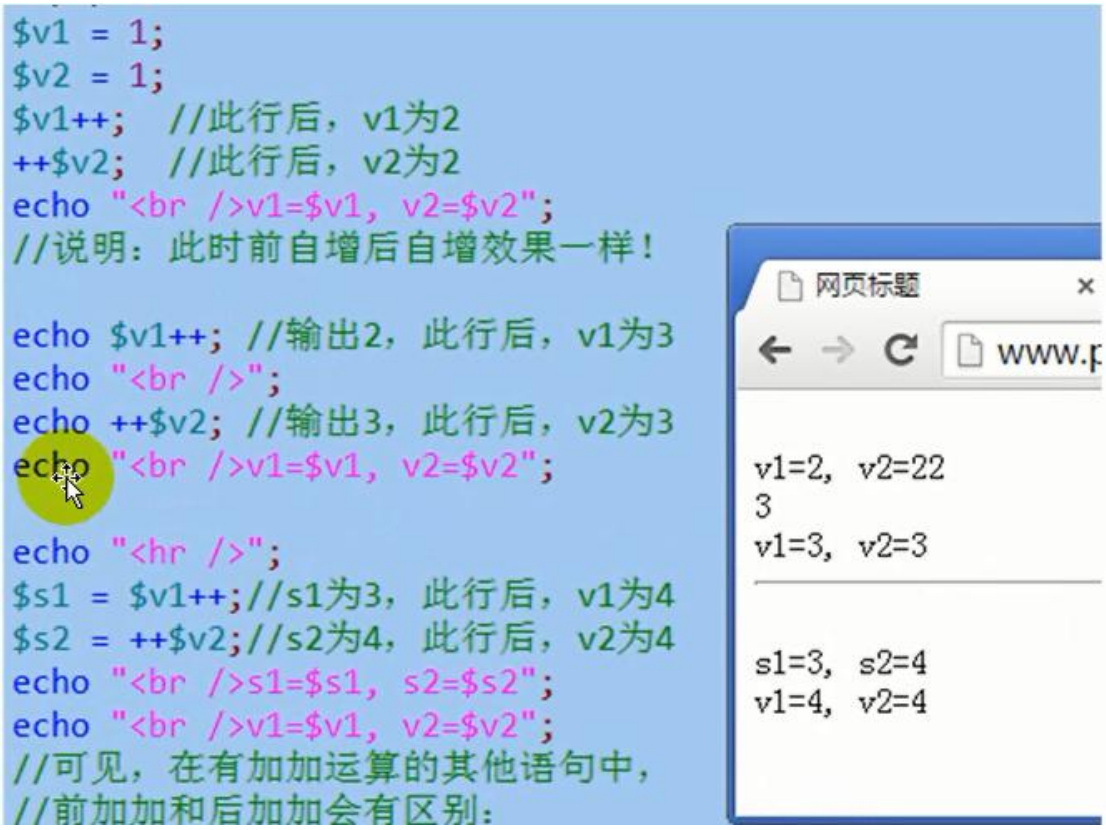
- 常规：对数字进行自加 1 或自减 1。
- 字符串： 只能自增，且自增的效果就是“下一个字符”：
- 布尔值递增递减无效
- null 递减无效，递增结果为 1

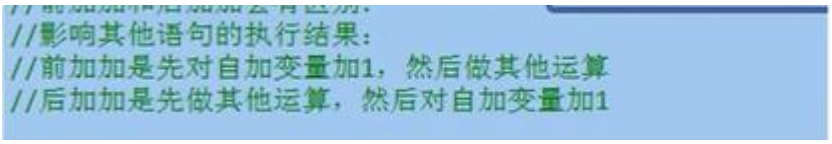
字符串自增的例子：

```
13 <?php
14 $s1 = "a";
15 $s2 = "A";
16 $s3 = "abc";
17 $s4 = "xyz";
18 $s5 = "xyz";
19 $s6 = "zzz";
20 $s7 = "abc9";
21 $s8 = "9z";
22 $s1++;$s2++;$s3++;$s4++;$s5++;$s6++;$s7++;$s8++;
23 echo "$s1, $s2, $s3, $s4, $s5, $s6, $s7, $s8";
24 ?>
```



前自增和后自增的区别（自减类似）





通常，在循环中，推荐使用前加加

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）。
- 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

- 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

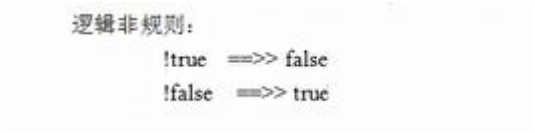
  - 对两边的字符串，一个一个从前往后取出字符并进行比较，谁“先大”，结果就是它大

```
“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”1”
```

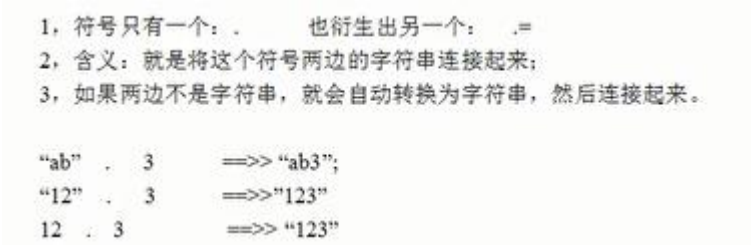
逻辑运算符

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

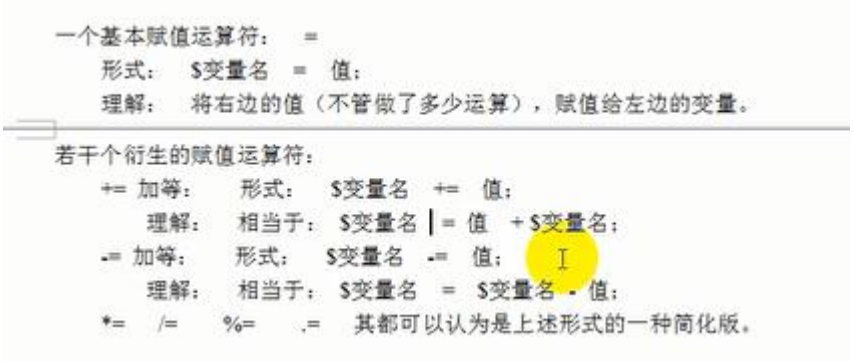
基本运算规则（真值表）



字符串运算符



赋值运算符



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“不及格”
```

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
```

按位异的规则是：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）。

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

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。

示例图示（只用 8 个位来演示）：

$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			

可见，结果为：$2^5 + 2^3 = 32 + 8 = 40$

```
22 $v1 = 10 << 2;
23 echo "<br />v1 = $v1";
```

v1 = 40