PART7

- 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

```php
<?php
header('Content-type:text/html; charset=utf-8');

$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";
?>
```

b, B, abd, xza, xzaa, aaaa, abd0, 10a

- 前自增和后自增的区别（自减类似）

```php

$v1 = 1;
$v2 = 1;
$v1++; //后自增
++$v2; //前自增
//此时结果都一样，都是2;

echo $v1++;//输出2，此行后V1=2
echo "<br/>";
echo ++$v2;//输出3，此行后V1=3
echo "<br/>v1 = $v1, v2 = $v2";

echo "<br/>";
echo "<hr/>";

$s1 = $v1++;//s1为3，此行后v1为4
$s2 = ++$v2;//s1为4，此行后v2为4
echo "<br/>s1=$s1, s2= $s2";
echo "<br/>v1=$v1, v2=$v2";
//可见，在有++运算的其他语句中
//前++和后++有区别，会影响其他语句的执行结果
//前++是先对自加变量+1，然后做其他运算；而后++相反
```

```
2
3
v1 = 3, v2 = 3
```

```
s1=3, s2= 4
v1=4, v2=4
```

通常，我们在循环中，推荐使用前加加，比如：
for($i = 1; $i < 10000; ++$i){ ..... }

```php
$t1 = microtime(true);//获得当前时间，精确到万分之一秒
for($i = 1; $i <100000000; ++$i){

}

$t2 = microtime(true);//获得当前时间，精确到万分之一秒
for($i = 1; $i <100000000; $i++){

}

$t3 = microtime(true);//获得当前时间，精确到万分之一秒
echo "<p>前++耗时：" . ($t2-$t1);
echo "<p>后++耗时：" . ($t3-$t2);
```

```
前++耗时：3.5384171009064
后++耗时：6.1585619449615
```

- 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。

- 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。
“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”
“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”：

```php
$n1 = 3;
$n2 = 4;


//if判断语句写法1：
if( $n1 > $n2  &&  f1($n1, $n2)  > 20 ) {
    //这里1
}
else{
    //这里2
}
//if判断语句写法2：
if( f1($n1, $n2)  > 20  &&  $n1 > $n2 ) {
    //这里op1
}
else{
    //这里op2
}
//写法1和写法2，最终计算结果是一样的！
//但写法1会具有优势：它有时候可能不需要进行“复杂”计算
//就可以得到判断结果，这就是“短路”现象
//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


//函数f1表示一个复杂的计算：
```

结论：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：
结论：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 = 90;          //分数
$valuation = $score >= 60？ “及格” ： “不及格” ；     //结果为“及格”

$score = 50;          //分数
$valuation = $score >= 60？ “及格” ： “不及格” ；     //结果为“不及格”

$score = 50;          //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）。
示例：

```php
$v1 = 10 & 20;
$v2 = 18 & 19;
echo $v1 ,$v2;
```

结果：

```
018
```

- 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。
示例：

```php
echo "<br/>";
echo "<hr/>";
$h1 = 10 << 3;
echo $h1;
```

```
80
```