第 5 章　运算符与流程控制

本章主题：

- 运算符。
- 运算符的优先级与结合性。
- 流程控制语句。
- 包含文件。

运算符是一种可以对一个或多个操作数进行运算并产生结果的符号。运算符体现了一种语言所具有的对数据的直接处理能力。一般情况下，一种运算符的操作数要有特定的数据类型，但很多情况下，PHP 会根据上下文自动将某种类型的数据转换成所需的类型。

程序的执行流程涉及程序的 3 种基本结构：顺序结构、选择结构和循环结构。从总体上看，程序代码是按顺序执行的，即按语句出现的先后次序依次执行语句，只有前面的语句执行完了才能执行后面的语句。选择结构是指在程序执行过程中，可以在两段代码中选择一段来执行，或者对一段代码是否执行进行选择。循环结构是指在程序执行过程中，可以对一段代码重复执行若干次。选择结构和循环结构都需要由一些特定的语句来实现。流程控制语句还包括跳转语句和包含文件语句。

本章首先介绍 PHP 的各种运算符及其使用，然后介绍控制流程的各种语句，包括实现选择结构的语句、实现循环结构的语句、跳转语句以及包含文件语句。

5.1　运算符

根据操作数的多少，运算符可分为单目运算符、双目运算符和三目运算符。只有一个操作数的运算符称为单目运算符，如果运算符出现在操作数之前，可称为前置运算符；如果运算符出现在操作数后面，可称为后置运算符。大多数运算符需要两个操作数且总是出现在两个操作数之间，称为双目运算符。需要 3 个操作数的运算符称为三目运算符，大多数编程语言支持三目条件运算符。

根据运算符的功能特点，PHP 运算符又可分为算术运算符、字符串运算符、比较运算符、逻辑运算符、位运算符、赋值运算符、三目条件运算符等。

5.1.1　算术运算符

算术运算符包括负号(－)、加(＋)、减(－)、乘(＊)、除(/)、求余(％)、指数运算(＊＊)、增 1(＋＋)和减 1(－－)等，如表 5-1 所示。

表 5-1　算术运算符

运　算　符	含　　义	例　　子
－	负号(单目)	－$x
＋	加法(双目)	$x＋$y
－	减法(双目)	$x－$y
＊	乘法(双目)	$x＊ $y

续表

运　算　符	含　　义	例　　子
/	除法(双目)	$x/$y
%	求余(双目)	$x%$y
**	指数运算(双目)	$x**$y
++	增1(单目)	++$x,$x++
--	减1(单目)	--$x,$x--

除法运算一般返回浮点数。如果两个操作数都是整数(或字符串转换成的整数),并且正好能整除,此时它返回一个整数。

求余运算符(%)适合两个整数的求余运算,运算结果的符号(正负号)与被除数的符号相同。可以使用 fmod 函数进行两个浮点数的求余运算。

在做算术运算时,如果出现非数值型操作数,则 PHP 会尝试将其转换成数值型数据,然后再进行算术运算。如果转换不了,通常会抛出一个 Fatal 错误(TypeError)。

在 PHP 中,算术运算存在以下规则。

(1) 进行算术除法运算(包括求余运算%)时,除数不能为零,否则会抛出一个 Fatal 错误(DivisionByZeroError)。

(2) 在进行整数算术运算时,如果运算结果超出整型数的取值范围(溢出),那么运算结果自动转换成浮点型。

(3) 在进行浮点数算术运算时,如果运算结果超出范围(上溢),则结果为无限值(正无穷或负无穷)。预定义常量 INF 表示正无穷。利用 is_infinite 函数可以测试一个值是否为无限值。

(4) 浮点数运算的结果也可能是 NaN(Not a Number),例如,两个无穷大相除、一个正无穷减去一个正无穷、0 乘以无穷大等。NAN 是一个表示该值的预定义常量。利用 is_nan 函数可以测试一个值是否为 NaN。

【例 5-1】　算术运算符。代码如下:

```
1. //除法运算
2. var_dump(12/4);
3. var_dump(12/5);
4. //求余运算
5. $a=12;
6. $b=5;
7. echo $a %$b,' ', $a %-$b, ' ', -$a %$b, ' ', -$a %-$b, '<br/>';
8. //指数运算
9. $r=3;
10. echo M_PI*$r**2, "<br />";
11. //增 1、减 1 运算
12. $x=10;
13. echo $x++, ' ', $x, '<br/>';
14. $x=10;
15. echo ++$x, ' ', $x, '<br/>';
```

下面是代码运行的输出结果:

```
int(3) float(2.4) 2 2 -2 -2
```

```
28.274333882308
10 11
11 11
```

第 10 行代码用到了 PHP 的一个预定义常量 M_PI，表示圆周率的近似值。

5.1.2 字符串运算符

字符串运算符是指字符串连接运算符(.)，用于将两个字符串连接成一个新的字符串返回。在做字符串连接运算时，如果出现非字符串操作数，那么 PHP 会尝试将其转换成字符串，然后再进行连接运算。

【例 5-2】 字符串运算符。代码如下：

```
1. $a="abc";
2. $b="xyz";
3. echo $a." 123 ".$b."<br />";
4.
5. $c=123;
6. $d=45.6;
7. echo "$c+$d=".$c+$d."<br />";
8. echo $c.$d."<br />";
```

下面是代码运行的输出结果：

```
abc 123 xyz
123+45.6=168.6
12345.6
```

该例中，第 7 行代码涉及字符串连接和算术加的混合运算。自 PHP 8 开始，算术运算符的优先级要高于字符串连接运算符。

5.1.3 比较运算符

比较运算符用于比较两个数据的大小，包括>、>=、<、<=、==、!=、<=>、===、!==等，如表 5-2 所示。

表 5-2 比较运算符

运 算 符	名 称	例 子
>	大于	$x>$y
>=	大于或等于	$x>=$y
<	小于	$x<$y
<=	小于或等于	$x<=$y
==	相等	$x==$y
!=或<>	不相等	$x!=$y，$x<>$y
<=>	太空船运算符	$x<=>$y
===	全等(类型和值均相同)	$x ===$y
!==	不全等	$x !==$y

其中，太空船运算符“$x<=>$y”是一种组合比较符，当$x小于、等于、大于$y时，分别返回一个小于、等于、大于0的int值。除太空船运算符外，其他比较运算符的运算结果都是布尔型的。

使用全等比较符(===)或不全等比较符(!==)比较两个数据时不涉及数据类型的转换，因为此时两个操作数的类型和值都要进行比较。仅当两个操作数的类型和值均相同时，全等比较(===)才成立。只要两个操作数的类型不同，或者它们的值不相等，不全等比较(!==)就成立。

当使用其他运算符比较两个数据时，如果两个操作数的类型不同，就会进行自动类型转换，使两个操作数具有相同类型，然后再进行比较。对两个标量类型数据(包括null)进行比较运算时，有关数据类型转换的规则如下。

(1) 如果一个操作数是null，另一个操作数是字符串，则先将null转换成空串，然后再进行比较；否则，进行下一步。

(2) 如果一个操作数是null，另一个操作数为非字符串，则两个操作数先都转换成布尔型，然后再进行比较。这里，false<true；否则，进行下一步。

(3) 如果一个操作数是布尔型，则将另一个操作数也转换成布尔型，然后再进行比较；否则，进行下一步。

(4) 如果两个操作数都是数值型(包括数字字符串)，就按数值比较大小；否则，进行下一步。

(5) 按字符串比较两个操作数的大小。如果其中一个操作数为数值，则先将其转换成字符串。

【例5-3】　比较运算符的使用。代码如下：

```
1. var_dump(null<"0");          // true(null转换成空串)
2. var_dump(null<0);            // false(null和0都转换成false)
3. var_dump(false<-1);          // true(-1转换成true)
4. var_dump(false<=>-1);        // -1
5. var_dump("10"=="0o10");      // false(按数值比较)
6. var_dump("10"<=>"0o10");     // 1
7. var_dump("10" =="1e1");      // true(按数值比较)
8. var_dump("10" =="10a");      // false(按字符串比较)
9. var_dump("10"<=>"10a");      // -1
10. var_dump("10" ==="1e1");    // false(类型相同,值不同)
```

5.1.4　逻辑运算符

逻辑运算符包括!、&&、||、xor、and和or。逻辑运算符的操作数应该是布尔型的，否则会自动进行类型转换。逻辑运算的结果是布尔型的。逻辑运算符及其含义如表5-3所示。

表5-3　逻辑运算符

运　算　符	含　义	例　子
!	逻辑非	!$x
&&	逻辑与	$x && $y
||	逻辑或	$x || $y
xor	逻辑异或	$x xor $y
and	逻辑与	$x and $y
or	逻辑或	$x or $y

逻辑非(!)是单目运算符，其运算结果与操作数的值正好相反。操作数为true，结果为false；操作

数为 false，结果为 true。

逻辑与运算符(&&、and)具有“并且”的含义，只有当两个操作数的值均为 true 时，运算结果才为 true；否则，运算结果为 false。

逻辑或运算符(||、or)具有“或者”的含义，两个操作数中，只要有一个为 true，则运算结果就为 true；否则，运算结果为 false。

逻辑异或运算符(xor)具有“唯一”和“或者”两重含义，两个操作数中，当有一个为 true，而另一个不为 true 时，运算结果为 true；否则，运算结果为 false。

逻辑运算符 && 和 and 都是快速逻辑与，逻辑运算符||和 or 都是快速逻辑或。即如果左操作数已经能决定运算结果，则右操作数将不再计算。它们之间的区别只是优先级不同，&& 的优先级高于 and，||的优先级高于 or。

在 PHP 中，逻辑运算符 and、or 和 xor 的优先级比赋值运算符的还要低。

【例 5-4】 逻辑运算符。代码如下：

```
 1. $a=10;
 2. $b=8;
 3. $c=$a>=10||++$b>8;
 4. var_dump($c);                    // true
 5. var_dump($b);                    // 8
 6.
 7. $d=$a>=10&&++$b>9;
 8. var_dump($d);                    // false
 9. var_dump($b);                    // 9
10.
11. $e=($a>=10 and $b>9);
12. $f=$a>=10 and $b>9;
13. var_dump($e);                    // false
14. var_dump($f);                    // true
```

该例中，第 3 行代码的逻辑或(||)运算符的左操作数($a>=10)的值是逻辑真 true，所以右操作数将不再计算，变量 $b 的值仍为 8。

第 7 行代码中，逻辑与(&&)运算符的左右操作数依次计算，左操作数为逻辑真 true，右操作数为逻辑假 false，逻辑与的运算结果为 false。在计算右操作数时，变量 $b 的值变为 9。

在第 12 行代码中，赋值运算符(=)的优先级高于逻辑与运算符(and)，所以先做赋值(=)运算，结果变量 $f 为逻辑真 true，然后再做逻辑与(and)运算，但运算结果没有保存，被丢弃了。

5.1.5 位运算符

位运算符包括～、&、|、^、<<、>>，如表 5-4 所示。位运算符的操作数应该是整型，否则会自动进行类型转换。位运算符将一个整型值当作一系列的二进制位来处理。位运算的结果是整型。

表 5-4 位运算符

运 算 符	含 义	例 子
～	按位取反(单目)	～$x
&	按位与	$x & $y
|	按位或	$x | $y
^	按位异或	$x ^ $y

续表

运　算　符	含　　义	例　　子
<<	左位移	$x << $y
>>	右位移	$x >> $y

前面4个运算符(～、&、|、^)也称为位逻辑运算符。原则上,位逻辑运算与前面介绍的逻辑运算的运算规则是相同的,只是逻辑运算符的运算对象是布尔型数据(true或false),而位逻辑运算符的运算对象是操作数(整数)内部的二进制位数据(0或1)。

注意:如果左右操作数都是字符串,则位逻辑运算符将对字符的ASCII值进行操作。

后面两个运算符(<<、>>)也称为位移运算符。向任何方向移出去的位都被丢弃。左移时右侧以零填充,符号位被移走意味着正负号不被保留。右移时左侧以符号位填充,意味着正负号被保留。

【例5-5】 位运算符的使用。代码如下:

```
1. $a=-10;
2. $b=10;
3. echo ～$a, " ", $a&$b, " ", $a|$b, " ", $a^$b, "<br />";
4.
5. $x=-2;
6. $y=2;
7. echo $x<<$y, " ", $x>>$y;
```

下面是代码运行的输出结果:

```
9 2 -2 -4
-8 -1
```

该例中,第3行代码涉及4个位逻辑运算,第7行代码涉及两个位移运算。为使运算过程更加清晰,下面给出了各操作数及运算结果的内部二进制表示:

```
$a:-10                  11111111111111111111111111110110
$b:10                   00000000000000000000000000001010
～$a:9                  00000000000000000000000000001001
$a&$b:2                 00000000000000000000000000000010
$a|$b:-2                11111111111111111111111111111110
$a^$b:-4                11111111111111111111111111111100
$x:-2                   11111111111111111111111111111110
$y:2                    00000000000000000000000000000010
$x<<$y:-8               11111111111111111111111111111000
$x>>$y:-1               11111111111111111111111111111111
```

其中,假定整数采用32位的二进制补码表示。

5.1.6　赋值运算符

赋值运算符包括简单赋值运算符和组合赋值运算符。

1. 简单赋值运算符

简单赋值运算符(=)的语法格式如下:

```
<变量>=<表达式>
```

其功能是计算赋值运算符右端表达式的值,并将其赋给运算符左端变量。该值同时作为整个赋值

运算表达式的运算结果。

2. 组合赋值运算符

组合赋值运算符(<op>=)将计算和赋值两种功能组合在一起,其语法格式如下:

```
<变量><op>=<表达式>
```

其功能是将变量的值与表达式的值按指定的运算符 op 进行计算,然后再把计算的结果重新赋给变量。这里 op 包括双目算术运算符、字符串运算符、双目位运算符。

【例 5-6】 赋值运算符。代码如下:

```
1. $a=($b=4) +5;
2. echo $a, " ", $b, "<br />";
3.
4. $x=3;
5. $x+=5;
6. $y="Hello ";
7. $y.="There!";
8. echo $x, " ", $y;
```

下面是代码运行的输出结果:

```
9 4
8 Hello There!
```

5.1.7 其他运算符

这里介绍三目条件运算符(?:)、Null 联合运算符(??)和类型运算符 instanceof 等。

1. 三目条件运算符

三目条件运算符?:的操作数有三个,其语法格式如下:

```
<op1>?<op2>:<op3>
```

其中,op1 应该是布尔型,否则会自动进行类型转换。如果 op1 为 true,则 op2 作为表达式的结果;如果 op1 为 false,则 op3 作为表达式的结果。在此,op2 和 op3 只选择其一进行计算。

【例 5-7】 三目条件运算符。代码如下:

```
1. $a=1;
2. $b="apple";
3. $c=true ? $b : ++$a;
4. echo $c," ", $a, "<br />";
5. echo (true ? 'true' : false) ? 't' : 'f';
```

下面是代码运行的输出结果:

```
apple 1
t
```

该例中,第 3 行代码的三目条件运算表达式中,第 1 个操作数的值为 true,所以只计算第 2 个操作数 $b 的值作为表达式的值,第 3 个操作数++$a 并没有计算,所以变量 $s 的值并没有变化。

第 5 行代码涉及三目条件运算符的嵌套。自 PHP 8 开始,三目条件运算符不具有结合性,所以需要用圆括号明确其计算次序,否则会产生 Fatal 错误。

2. Null 联合运算符

Null 联合运算符“??”实际上是特定情况下三目条件运算符的简化版，其语法格式如下：

```
<expr1>?? <expr2>
```

如果 expr1 的值不为 null(或者说已经设置)，那么运算结果为 expr1；否者，运算结果为 expr2。在检测 expr1 值时，不会产生任何 Notice 或 Warning 错误。其在功能上与下面三目条件运算表达式一致：

```
isset(<expr1>) ? <expr1>: <expr2>
```

3. 错误控制运算符

错误控制运算符@可以放置在一个表达式之前，这样该表达式产生的任何 Notice 错误或 Warning 错误将被抑制，即这些错误不会被报告。

4. 类型运算符

类型运算符 instanceof 用于检测一个对象是否为某个特定类或其子类的实例，其语法格式如下：

```
<op1>instanceof <op2>
```

其中，op1 应该是一个对象，op2 表示某个类类型或接口类型。如果对象 op1 是类 op2 或其子类的实例，或者是实现接口 op2 的某个类的实例，则表达式返回 true，否则返回 false。

【例 5-8】　类型运算符。代码如下：

```
1. class SuperClass {}
2. class SubClass extends SuperClass {}
3. class OtherClass {}
4.
5. $o=new SubClass();
6.
7. var_dump($o instanceof SubClass);
8. var_dump($o instanceof SuperClass);
9. var_dump($o instanceof OtherClass);
```

下面是代码运行的结果：

```
bool(true) bool(true) bool(false)
```

该例中，SuperClass 是 SubClass 的超类，而 OtherClass 不是 SubClass 的超类。SubClass 类的实例可以被看作 SuperClass 类的对象，但不能被看作 OtherClass 类的对象。

5.2　表达式

表达式是由运算符和操作数按一定的语法规则连接起来的式子。前面已经对运算符做了较为全面的介绍，包括算术运算符、关系运算符、逻辑运算符、位运算符、三目条件运算符以及赋值运算符等。操作数包括文字、常量、变量、函数调用、数组元素访问等。

在此，各种操作数本身也是表达式。所以表达式的这个定义是递归的，即在表达式的定义中又用到了表达式自身。

当一个表达式包含多个运算时，弄清楚它们的计算次序是非常重要的，这决定着表达式的计算结果。例如，表达式 x+5-(x=10)+x*2，是先计算 x+5 还是先计算 x*2？其中的 x 是取原先的值还

是取 10?

运算符的优先级和结合性影响着表达式的计算次序。当一个操作数同时为前后两个运算符的操作数时，如果前面运算符的优先级高于后面运算符的优先级，则该操作数作为前面运算符的操作数先进行运算，运算结果作为后面运算符的操作数；如果后面运算符的优先级高于前面运算符的优先级，则该操作数作为后面运算符的操作数先进行运算，运算结果作为前面运算符的操作数；如果前后两个运算符的优先级相同，则要看运算符的结合性。如果是左结合(从左到右)，则该操作数作为前面运算符的操作数先进行运算，运算结果作为后面运算符的操作数；如果是右结合(从右到左)，则该操作数作为后面运算符的操作数先进行运算，运算结果作为前面运算符的操作数。例如：

```
x+y*z          //相当于 x+(y * z),*优先级高于+
x=y-z          //相当于 x=(y-z),-优先级高于=
x+y-z          //相当于(x+y) -z,同级左结合
x=y=13         //相当于 x=(y=13),同级右结合
```

表 5-5 列出了 PHP 运算符的优先级和结合性。其中，各运算符按优先级降序排列，优先级为 1 的运算符优先级最高，同一行中的运算符的优先级相同。

表 5-5 运算符的优先级和结合性

优先级	运 算 符	结 合 性
1	new	—
2	[]	左
3	**	右
4	++、--、-、~、(int)、(float)、(string)、(array)、(object)、(bool)、@	—
5	instanceof	左
6	!	—
7	*、/、%	左
8	+、-	左
9	<<、>>	左
10	.	左
11	<、<=、>、>=	—
12	==、!=(<>)、<=>、===、!==	—
13	&	左
14	^	左
15	|	左
16	&&	左
17	||	左
18	??	右
19	? :	—
20	=、+=、-=、*=、/=、%=、.=、&=、|=、^=、<<=、>>=	右

续表

优先级	运　算　符	结　合　性
21	and	左
22	xor	左
23	or	左
24	,	左

从表中可以发现，有些运算符没有规定结合性，如比较运算符等。当出现前后两个运算符具有相同的优先级但没有规定结合性时，将无法确定它们计算的先后次序，所以这种情况是非法的。例如，表达式 1＜＄x＜＝10 是非法的，因为运算符＜和＜＝的优先级相同但没有规定结合性。而表达式 1 ＜＝ ＄x ＝＝ 1 是合法的，虽然运算符＜＝和＝＝都没有规定结合性，但它们的优先级不同。

总之，表达式的计算按从左到右并尊重运算符的优先级和结合性的原则进行。表达式在从左到右的计算过程中，当前运算符的操作数首先被计算（左操作数先于右操作数被计算），然后再判断当前运算是马上执行还是暂缓执行；如果当前运算符的优先级高于后面运算符的优先级，或者前后两个运算符优先级相同但为左结合，则当前运算就可以马上执行，运算结果将作为新的操作数参与表达式接下来的计算过程；如果当前运算符的优先级低于后面运算符的优先级，或者前后两个运算符优先级相同但为右结合，则暂缓执行当前运算，转而考虑下一个运算。

另外，有些操作数本身就是一个由“()”括起来的子表达式，对这些操作数的计算同样需要遵循上述原则和过程。下面通过例子说明表达式的计算过程。

假设变量 ＄x 已经由下面的语句定义：

```
$x =15;
```

表达式 ＄x＋5－(＄x＝10)＋＄x＊2 的计算过程如下。

步骤 1：做加法运算，表达式变为“20 －(＄x＝10)＋＄x＊2”。

步骤 2：计算子表达式，表达式变为“20－10＋＄x＊2”，其中，变量 ＄x 的值变为 10。

步骤 3：做减法运算，表达式变为“10＋＄x＊2”。

步骤 4：做乘法运算，表达式变为“10＋20”。

步骤 5：做加法运算，得到表达式的计算结果为整数 30。

这里，步骤 1 做加法运算时，变量 ＄x 取原先的值 15。步骤 2 计算子表达式，其计算过程与一般表达式的计算过程相同。该子表达式的计算除了返回结果值 10 之外，还将变量 ＄x 置为 10，所以后面再访问变量 ＄x 时将使用该新值。

5.3　流程控制

本节介绍控制程序执行流程的相关语句，包括支持选择结构的语句、支持循环结构的语句以及若干跳转语句。

5.3.1　语句与语句块

PHP 脚本是语句的集合。一条 PHP 语句可以是赋值语句、表达式语句、条件语句、循环语句、函数调用等，甚至可以是空语句。一条语句以“;”结尾，它是 PHP 语句的终止符号。