第三章 操作符

by flamephoenix

一、算术操作符 : +(加)、-(减)、*(乘)、/(除)、**(乘幂)、%(取余)、-(单目负)
(1)乘幂的基数不能为负，如 (-5) ** 2.5 # error;
(2)乘幂结果不能超出计算机表示的限制，如10 ** 999999 # error
(3)取余的操作数如不是整数，四舍五入成整数后运算；运算符右侧不能为零
(4)单目负可用于变量： - $y ; # 等效于 $y * -1
二、整数比较操作符

Table 3.1. 整数比较操作符

操作符	描述
<	小于
>	大于
==	等于
<=	小于等于
>=	大于等于
!=	不等于
<=>	比较，返回 1, 0, or -1

操作符<=>结果为:
0 - 两个值相等
1 - 第一个值大
1 - 第二个值大
三、字符串比较操作符

Table 3.2. 字符串比较操作符

操作符	描述	
lt	小于	
gt	大于	
eq	等于	

le	小于等于	
ge	大于等于	
ne	不等于	
cmp	比较，返回 1, 0, or -1	

四、逻辑操作符

逻辑或：$a || $b 或 $a or $b

逻辑与：$a && $b 或 $a and $b

逻辑非：! $a 或 not $a

逻辑异或：$a xor $b

五、位操作符

位与：&

位或：|

位非：~

位异或：^

左移：$x << 1

右移：$x >> 2

注：不要将&用于负整数，因为PERL将会把它们转化为无符号数。

六、赋值操作符

Table 3.3. 赋值操作符

操作符	描述
=	Assignment only
+=	Addition and assignment
-=	Subtraction and assignment
*=	Multiplication and assignment
/=	Division and assignment
%=	Remainder and assignment
**=	Exponentiation and assignment
&=	Bitwise AND and assignment
|=	Bitwise OR and assignment
^=	Bitwise XOR and assignment

Table 3.4. 赋值操作符例子

表达式	等效表达式
$a = 1;	none (basic assignment)
$a -	$a = $a -

= 1;	1;
$a *= 2;	$a = $a * 2;
$a /= 2;	$a = $a / 2;
$a %= 2;	$a = $a % 2;
$a **= 2;	$a = $a ** 2;
$a &= 2;	$a = $a & 2;
$a |= 2;	$a = $a | 2;
$a ^= 2;	$a = $a ^ 2;

.=可在一个赋值语句中出现多次，如:

```
$value1 = $value2 = "a string";
```

.=作为子表达式

```
($a = $b) += 3;
```

等价于

```
$a = $b;
$a += 3;
```

但建议不要使用这种方式。

七、自增自减操作符 : ++、--(与C++中的用法相同)

.不要在变量两边都使用此种操作符: ++$var-- # error

.不要在变量自增/减后在同一表达式中再次使用: $var2 = $var1 + ++$var1; # error

.在PERL中++可用于字符串，但当结尾字符为'z'、'Z'、'9'时进位，如:

```
$stringvar = "abc";
$stringvar++; # $stringvar contains "abd" now

$stringvar = "aBC";
$stringvar++; # $stringvar contains "aBD" now

$stringvar = "abz";
$stringvar++; # $stringvar now contains "aca"

$stringvar = "AGZZZ";
$stringvar++; # $stringvar now contains "AHAAA"

$stringvar = "ab4";
$stringvar++; # $stringvar now contains "ab5"

$stringvar = "bc999";
$stringvar++; # $stringvar now contains "bd000"
```

.不要使用--，PERL将先将字符串转换为数字再进行自减

```
$stringvar = "abc";
$stringvar--; # $stringvar = -1 now
```

.如果字符串中含有非字母且非数字的字符，或数字位于字母中，则经过++运算前值转换为数字零，因此结果为1，如:

```
$stringvar = "ab*c";
$stringvar++;
$stringvar = "ab5c";
```

```
$stringvar++;
```

八、字符串联结和重复操作符

联接: .

重复: x

联接且赋值(类似+=): .=

例:

```
$newstring = "potato" . "head";
$newstring = "t" x 5;
$a = "be";
$a .= "witched"; # $a is now "bewitched"
```

九、逗号操作符

其前面的表达式先进行运算，如:

```
$var1 += 1, $var2 = $var1;
```

等价于

```
$var1 += 1;
$var2 = $var1;
```

使用此操作符的唯一理由是提高程序的可读性，将关系密切的两个表达式结合在一起，如:

```
$val = 26;
$result = (++$val, $val + 5); # $result = 32
```

注意如果此处没有括号则意义不同:

```
$val = 26;
$result = ++$val, $val + 5; # $result = 27
```

十、条件操作符

与C中类似，条件?值1:值2，当条件为真时取值1，为假时取值2，如:

```
$result = $var == 0 ? 14 : 7;
$result = 43 + ($divisor == 0 ? 0 : $dividend / $divisor);
```

PERL 5中，还可以在赋值式左边使用条件操作符来选择被赋值的变量，如:

```
$condvar == 43 ? $var1 : $var2 = 14;
$condvar == 43 ? $var1 = 14 : $var2 = 14;
```

十一、操作符的次序

Table 3.6. 操作符次序

操作符	描述
++, --	自增，自减
-, ~, !	单目
**	乘方
=~, !~	模式匹配
*, /, %, x	乘，除，取余，重复
+, -, .	加，减，联接
<<, >>	移位
-e, -r, etc.	文件状态
<, <=, >, >=, lt, le, gt, ge	不等比较
==, !=, <=>, eq, ne, cmp	相等比较
&	位与
|, ^	位或，位异或
&&	逻辑与
||	逻辑或
..	列表范围

? and :	条件操作符
=, +=, -=, *=,	赋值
and so on	
,	逗号操作符
not	Low-precedence logical NOT
and	Low-precedence logical AND
or, xor	Low-precedence logical OR and XOR

.操作符结合性(associativity):

Table 3.7. 操作符结合性

操作符	结合性
++, --	无
-, ~, !	Right-to-left
**	Right-to-left
=~, !~	Left-to-right
*, /, %, x	Left-to-right
+, -, .	Left-to-right
<<, >>	Left-to-right
-e, -r,	无
<, <=, >, >=, lt, le, gt, ge	Left-to-right
==, !=, <=>, eq, ne, cmp	Left-to-right
&	Left-to-right
|, ^	Left-to-right
&&	Left-to-right
||	Left-to-right
..	Left-to-right
? and :	Right-to-left
=, +=, -=, *=,	Right-to-left
and so on	
,	Left-to-right
not	Left-to-right
and	Left-to-right
or, xor	Left-to-right

建议:

1、当你不确定某操作符是否先执行时，一定要用括号明确之。

2、用多行、空格等方式提高程序的可读性。