

一个准确但不礼貌的形容：**Perl** 是一种“面向操作符的语言”。正是操作符之间的互动以及它们的操作数给与了 **Perl** 表达力和力量。要理解 **Perl** 必须理解操作符和它们的行为。出于本次讨论，一个 **Perl** 操作符 的确切定义是：“它由一系列符号组成，用作语言语法的一部分”。每一个操作符接受零个或多个 操作数；这个定义是循环的，因为操作数也是由操作符操作的值。

对操作符最准确的定义是“`perlop`中的那个”，但它还是将一些操作符遗漏在 `perlsyn` 中并且把内置函数也算入其中。不要太执着于单个定义。

操作符特征

`perldoc perlop` 和 `perldoc perlsyn` 提供了大量有关 **Perl** 操作符行为的信息。即使如此，其中 没有 提到的部分对于理解这些操作符来说更为重要。这部分文档假定你对语言设计中的若干概念有着一定程度的熟悉。起初，这些概念也许听起来有些生硬，但是其实是直截了当的。

每一个操作符持有若干构成其行为的重要特征：操作数的个数，和其他操作符的关系，以及可能是用法。

优先级

某操作符的 优先级 有助 **Perl** 在对其在表达式中求值时做出决定。求值顺序从高到低。例如，因乘法的优先级较加法高， $7 + 7 * 10$ 求值得 77，而非 140。你可以用括号将子表达式分组使某操作符在其他操作符之前求值； $(7 + 7) * 10$ 确实 求值得 140，因为加法操作符已经成为一个整体，并且必须在乘法发生之前完全求值。

在平手的情况下——两个操作符拥有同样的优先级——则由其他因素（如词缀性 *fixity* 和结合性 *associativity*）来决定。

`perldoc perlop` 包含了一张优先级表。几乎没有人记得住这张表。掌控优先级的最佳途径就是让表达式尽量简单。另一种好方法就是在复杂的表达式中用括号来澄清优先级。如果你发现自己淹没在括号的海洋中，再请参考第一条规则。

结合性

某操作符的 结合性 决定了它是从左往右求值还是从右往左。加法是左结合的，因此 $2 + 3 + 4$ 先求出 $2 + 3$ ，然后再在此结果上加 4。指数操作是右结合的，因此 $2 ** 3 ** 4$ ，先进行 $3 ** 4$ 这部分运算，然后再求得 2 的 81 次方。

简化复杂的表达式并用括号来展示你的意图远比记住结合性表来得重要。虽说如此，记住结合性表的数学操作符部分还是值得的。

核心模块 `B::Deparse` 可以重写代码片段并如实展示 **Perl** 究竟是如何处理操作符优先级和结合性的；在某代码片段上运行 `perl -MO=Deparse,-p`。（`-p` 标志添加额外的分组括号使得求值顺序更为明显。）注意 **Perl** 的优化器会如前面的例子般简化数学操作，你可以用变量替代，就像 `$x ** $y ** $z`。

参数数量

操作符的 参数数量 就是该操作符所作用的操作数的个数。空元 操作符没有操作数。一元 操作符有一个操作数。二元 操作符有两个操作数。三元 操作符有三个操作数。列元 操作符对一个操作数列表进行操作。

除了大多数操作符都接受两个、多个或一个操作数这一事实，没有其他什么好的规则来决定一个操作符的参数个数。操作符的文档会把这些交待清楚。

举例说来，算术运算符是二元操作符，它们通常是左结合的。 $2 + 3 - 4$ 先对 $2 + 3$ 求值，加法和减法的优先级一致，但是它们是左结合并且是二元的，因此正确的求值顺序将最左端的操作符（+）应用到最左端的两个操作数（2 和 3）上，接着，将最右端的操作符（-）应用在第一个操作符的结果和最右端的操作数（4）上。

一个 Perl

新手疑问的常见来源就是嵌套表达式中的列元操作符（特别是函数调用）。请使用括号表明你的意图，并且留意如下代码中的问题：

```
# 可能是一段有缺陷的代码
say ( 1 + 2 + 3 ) * 4;
```

.....由于 Perl 5 高兴地将括号解释为后环缀（fixity）操作符并认为括号中的内容是 say 的参数，而并非为改变优先级而对表达式分组的环缀括号。换句话说，这段代码打印出 6 然后求值 say 的返回值乘以 4 后的结果。

词缀性

操作符的词缀性 就是它相对其操作数的位置：

中缀 操作符出现在操作数之间。大部分数学操作符是中缀操作符，例如 `$length * $width` 中的乘法操作符；

前缀 操作符出现于其操作符之前而 后缀 操作符出现于其操作符之后。这

些操作符通常是一元的，如数学上的负数符号（`-$x`）、布尔取反（`!$y`）以及后缀 自增（`$z++`）；

环缀 操作符包围其操作数。例子包括匿名哈希构造符（`{ ... }`）和引述操 作符（`qq[...]`）；

后环缀 操作符接在某些操作数之后并环绕其他部分，就向访问哈希或数组的元素那样（`$hash{ ... }` 和 `$array[...]`）。

操作符类型

Perl 无孔不入的上下文——特别是值上下文（`value_contexts`）——大大扩展了操作符 的行为。Perl

操作符对它们的操作数提供了值上下文。为给定的情况选择最合适的操作符要求你对所求值和所用值的类型都要有所理解。

数值操作符

数值操作符对其操作数强制数值上下文。它们由标准算术操作符构成：加（+）、减（-）、乘（*）、除（/）、指数（**）、取模（%）以及变种（+=、-=、*=、/=、**= 和 %=）外加前后缀自减（--）。

虽然自增操作符看上去像数值操作符，它有着特殊的字符串行为（`auto_increment_operator`）。

若干比较操作符对其操作数强制数值上下文。它们是数值等于（==）、数值不等于（!=）、大于（>）、小于（<）、大于等于（>=）、小于等于（<=），以及排序比较操作符（<=>）。

字符串操作符

字符串操作符对其操作数强制字符串上下文。它们由肯定和否定正则表达式绑定操作符（`=~` 和 `!~`，对应地），和拼接操作符（`.`）组成。

若干比较操作符对其操作数强制数值上下文。它们是字符串等（`eq`）、字符串不等于（`ne`）、大于（`gt`）、小于（`lt`）、大于等于（`ge`）、小于等于（`le`）以及字符串排序比较操作符（`cmp`）。

逻辑操作符

逻辑操作符在布尔上下文中处理其操作数。`&&` 和 `and` 操作符测试两边表达式是否都为逻辑真，`||` 和 `or` 操作符则测试两边的表达式是否有一个为真。这四个全部是中缀操作符。这四个全部执行短路测试：如果一个表达式的求值结果使得整个表达式为假，Perl 不会继续对其他表达式求值。单词形式比符号形式的优先级低。

已定义-或操作符，`//`，测试其操作数的定义性。不像 `||` 测试其操作数的为真性，如果操作数求值为数值零或空字符串，`//` 还是会返回真值。这对设置默认参数的值尤其有用。

```
sub name_pet
{
    my $name = shift // 'Fluffy';
    ...
}
```

三元条件操作符，`?:`，接受三个操作数。它对第一个操作符在布尔上下文中求值，并在结果为真时对第二个操作数求值，否则求值第三个：

```
my $truthiness = $value ? 'true' : 'false';
```

`!` 和 `not` 操作符返回其操作数的逻辑反值。`not` 的优先级比 `!` 低。它们是前缀操作符。

`xor` 操作符是中缀操作符，对其操作数进行异或操作。

按位操作符

按位操作符在位级别按数值形式处理它们的操作数。这些操作符在大多数 Perl 5 程序中并不常见。它们由左移（`<<`）、右移（`>>`）、按位与（`&`）、按位或（`|`）、以及按位异或（`^`）以及它们“就地”变种（`&=`、`|=`、`^=`、`<<=` 和 `>>=`）组成。

特殊操作符

自增操作符有一个特例。如果任何变量在数值上下文中使用过（`cached_coercions`），自增操作符将增加它的数值部分。如果这个变量明显地是一个字符串（且尚未在数值上下文中求值）则此字符串将会带进位地自增，因此 `a` 增加为 `b`、`zz` 增为 `aaa`，以及 `a9` 增为 `b0`。

```
my $num = 1;
my $str = 'a';

$num++;
$str++;
is( $num, 2, 'numeric autoincrement should stay numeric' );
is( $str, 'b', 'string autoincrement should stay string' );

no warnings 'numeric';
$num += $str;
$str++;
```

```

    is( $num, 2, 'adding $str to $num should add numeric value of $str'
);
    is( $str, 1, '... but $str should now autoincrement its numeric part'
);

```

重复操作符 (x) 是一个中缀操作符。在列表上下文中，它的行为按它的第一个操作数改变。当给与列表时，它的求值结果是一个重复列表，次数由第二个操作数给出。当给以一标量，它产生的值是将第一个操作数的字符串部分重复拼接到自身，次数由第二个操作数给出。在标量上下文中操作符总是产生重复拼接合适次数的字符串。

例如：

```

my @scheherazade = ('nights') x 1001;
my $calendar      = 'nights' x 1001;

is( @scheherazade, 1001, 'list repeated' );
is( length $calendar, 1001 * length 'nights', 'word repeated' );

my @schenolist    = 'nights' x 1001;
my $calscalar     = ('nights') x 1001;

is( @schenolist, 1, 'no lvalue list' );
is( length $calscalar, 1001 * length 'nights', 'word still repeated'
);

```

范围 操作符 (..) 是一个中缀操作符，它在列表上下文中产生一个列表：

```

my @cards = ( 2 .. 10, 'J', 'Q', 'K', 'A' );

```

它可以产生简单的、递增的范围（无论是整数还是自增字符串），但是它不能感知模式和更复杂的范围。

在布尔上下文中，范围操作符变为 翻斗 (flip-flop) 操作符。如果它的左操作数为假，这个操作符返回假值，然后当它的右操作符一直为真时返回真。因此你可以这样引述一封格式迂腐邮件的正文：

```

while (/Hello, $user/ .. /Sincerely,/)
{
    say "> $_";
}

```

逗号 操作符 (,) 是一个中缀操作符。在标量上下文中它先求值它的左操作数然后返回对其右操作数求值的结果。在列表上下文中，它按从左到右的顺序对两边的操作数求值。

胖逗号操作符 (=>) 的行为与此类似，除了它自动对用做其左操作数的裸字加上引号。（参见 hashes）。