第 3 章
运算符

在 Go 语言中，运算符(Operator)是一种特殊的符号，它表示应该执行何种计算。运算符作用的对象称为操作数(Operand)。

```
var a int = 1
var b int = 2
fmt.Println(a + b)                                    //"+"是运算符,变量 a 和 b 是操作数
```

用运算符和括号将操作数连接起来的、符合 Go 语法规范的式子，称为表达式(Expression)，如 a + b − 5。一个单独的常量或变量也是表达式。Go 语言的内置运算符有算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符和其他运算符。

3.1 算术运算符

Go 语言中一共有 9 个算术运算符，如表 3-1 所示。

表 3-1 算术运算符

运算符	类 别	意 义	示 例	说 明
+	一元	正号	+a	正号
+	二元	求和	a + b	计算 a 与 b 的和
−	一元	负号	−a	负号
−	二元	求差	a − b	计算 a 与 b 的差
*	二元	乘法	a * b	计算 a 与 b 的积
/	二元	除法	a / b	计算 a 除以 b 的商
%	二元	求余	a % b	计算 a 除以 b 的余数
++	一元	自增	a++	等价于 a = a + 1
−−	一元	自减	a−−	等价于 a = a−1

举例：

```
func main() {
    var a, b int = 5, 3
    fmt.Println("+a", +a)                              //输出+a 5
```

```
    fmt.Println("a + b", a+b)                    //输出 a + b 8
    fmt.Println("-a", -a)                        //输出-a -5
    fmt.Println("a - b", a-b)                    //输出 a - b 2
    fmt.Println("a * b", a*b)                    //输出 a * b 15
    fmt.Println("a / b", a/b)                    //输出 a / b 1
    fmt.Println("a % b", a%b)                    //输出 a % b 2
    a++
    fmt.Println("a", a)                          //输出 a 6
    b--
    fmt.Println("b", b)                          //输出 b 2
}
```

n % m 的计算结果是[0，m－1]。求余运算的一个应用场景是求一个多位数各个数位上的数字。

```
func main() {
    var num int = 351
    fmt.Println(num / 100)                      //得到 num 百位上的数字 3
    fmt.Println(num / 10 % 10)                  //得到 num 十位上的数字 5
    fmt.Println(num % 10)                       //得到 num 个数上的数字 1
}
```

两个整数相除的结果为整数，而不是浮点数，这与 C 语言相同，如 5 / 2 ＝ 2，而不是 2.5。表 3-2 给出了两个整数的除法与求余运算的 4 种情况。求余运算的结果，其符号与被除数的符号相同。

表 3-2　两个整数的除法与求余运算

x	y	x / y	x % y
5	3	1	2
－5	3	－1	－2
5	－3	－1	2
－5	－3	1	－2

3.2　关系运算符

Go 语言中一共有 6 个关系运算符，如表 3-3 所示。

表 3-3　关系运算符

运算符	意　义	示　例	说　明
＝＝	等于	a ＝＝ b	如果 a 等于 b，则结果为 true；否则为 false
!=	不等于	a != b	如果 a 不等于 b，则结果为 true；否则为 false
＜	小于	a ＜ b	如果 a 小于 b，则结果为 true；否则为 false
＜＝	小于或等于	a ＜＝ b	如果 a 小于或等于 b，则结果为 true；否则为 false

续表

运算符	意　义	示　例	说　明
>	大于	a > b	如果 a 大于 b,则结果为 true;否则为 false
>=	大于或等于	a >= b	如果 a 大于或等于 b,则结果为 true;否则为 false

举例:

```
func main() {
    var a, b int = 5, 3
    fmt.Println("a == b", a == b)                    //输出 a == b false
    fmt.Println("a != b", a != b)                    //输出 a != b true
    fmt.Println("a < b", a < b)                      //输出 a < b false
    fmt.Println("a <= b", a <= b)                    //输出 a <= b false
    fmt.Println("a > b", a > b)                      //输出 a > b true
    fmt.Println("a >= b", a >= b)                    //输出 a >= b true
}
```

3.3　逻辑运算符

Go 语言中一共有 3 个逻辑运算符,如表 3-4 所示。

表 3-4　逻辑运算符

运算符	示例	说　明
&&	a && b	逻辑与,如果 a 与 b 都为 true,则结果为 true;否则为 false
||	a || b	逻辑或,如果 a 与 b 都为 false,则结果为 false;否则为 true
!	!a	逻辑非,如果 a 为 false,则结果为 true;否则为 false

举例:

```
func main() {
    var a bool = true
    var b bool = false
    fmt.Println("a && b", a && b)                    //输出 a && b false
    fmt.Println("a || b", a || b)                    //输出 a || b true
    fmt.Println("!a", !a)                            //输出!a false
}
```

在逻辑表达式的求解过程中,并不是所有的表达式都会被执行。只有在必须执行下一个表达式才能确定其值时,才会执行这个表达式,这种特性叫作短路求值(Short-Circuit Evaluation)。计算 exp1 && exp2 时,如果 exp1 的值为 false,那么就不会计算 exp2 的值。因为不论 exp2 的值是 true 还是 false,整个表达式的值一定是 false,而不受 exp2 值的影响。同理,计算 exp1 || exp2 的值时,只要 exp1 的值为 true,就不会再计算 exp2 的值。

在某些情况下,利用短路求值特性可以编写出既简洁又高效的代码。假如有两个变量 a 和 b,想知道表达式 b / a 是否大于 0。考虑到 a 可能为 0,可以这样编写代码:

```
a != 0 && (b / a) > 0
```

当 a 为 0 时，a != 0 为 false，短路求值特性确保第二个表达式(b / a) > 0 不会被计算，从而避免引发程序异常。

3.4　位运算符

位运算符将操作数看作一个二进制数字序列，并对其进行逐位操作。Go 语言支持的位运算符如表 3-5 所示。

表 3-5　位运算符

运算符	示　例	意　义	说　明
&	a & b	按位与	对应位的与运算，只有两者都是 1，才为 1；否则为 0
|	a | b	按位或	对应位的或运算，只要有一个为 1，则为 1；否则为 0
^	^a	按位取反	将每一位都求反，如果为 0，则为 1；如果为 1，则为 0
^	a ^ b	按位异或	对应位的异或运算，如果两者不同，则为 1；否则为 0
&^	a &^ b	按位清零	b 的某位为 1 时，将 a 中的对应位清零，其他位不变
>>	a >> n	按位右移	将每一位都向右移动 n 位
<<	a << n	按位左移	将每一位都向左移动 n 位

举例：

```
func main() {
    var a uint8 = 12                    //12 的二进制形式 00001100
    var b uint8 = 10                    //10 的二进制形式 00001010
    fmt.Println("a & b", a&b)           //输出 a & b 8,8 的二进制形式 1000
    fmt.Println("a | b", a|b)           //输出 a | b 14,14 的二进制形式 1110
    fmt.Println("^a", ^a)               //输出^a 243,243 的二进制形式 11110011
    fmt.Println("a ^ b", a^b)           //输出 a ^ b 6,6 的二进制形式 110
    fmt.Println("a &^ b", a&^b)         //输出 a &^ b 4,4 的二进制形式 100
    fmt.Println("a >> 2", a>>2)         //输出 a >> 2 3,3 的二进制形式 11
    fmt.Println("a << 2", a<<2)         //输出 a << 2 48,48 的二进制形式 110000
}
```

3.5　赋值运算符

在 Go 语言中，“=”就是赋值运算符，其作用是将“=”右侧表达式的值赋给“=”左侧的变量，如 b = a + 2。在赋值运算符“=”的前面加上其他运算符，就构成了复合赋值运算符。如果在“=”的前面添加一个“+”运算符，就构成了复合赋值运算符“+=”。

```
a += 1 等价于 a = a + 1
a %= 3 等价于 a = a % 3
a ^= 5 等价于 a = a ^ 5
```

```
a *= 5 - 2 等价于 a = a * (5 - 2)          //注意,不是等价于 a = a * 5 - 2
```

算术运算符支持这种用法：

```
+=、-=、*=、/=和%=
```

举例：

```
func main() {
    var a int = 2
    a *= 3 - 1                              //等价于 a = a * (3 - 1),而不是 a = a * 3 - 1
    fmt.Println(a)                          //输出是 4,而不是 5
}
```

位运算符也支持这种用法：

```
&=、|=、^=、>>=和<<=
```

举例：

```
func main() {
    var a uint8 = 2                         //2 的二进制形式是 00000010
    a <<= 1                                 //等价于 a = a << 1
    fmt.Println("a =", a)                   //输出 a = 4
}
```

Go 语言的其他运算符有两个：一个是用于获取变量的存储地址 &；另一个是指针运算符 *。本书后面的章节再介绍这两个运算符。

3.6 运算符的优先级

求解一个表达式的值时，必须按照运算符的优先级(Preference)从高到低的顺序进行计算(括号除外)。表 3-6 按照优先级从高(级别 8)到低(级别 1)的顺序列出 Go 语言的部分运算符。

表 3-6 运算符的优先级

优先级	运 算 符	说 明
8	!、+x、-x、^x	逻辑非、正号、负号、按位取反
7	*、/、%、<<、>>、&、&^	乘、除、求余、左移、右移、按位与、按位清零
6	+、-、|、^	加、减、按位或、按位异或
5	==、!=、<、<=、>、>=	相等、不等、小于、小于或等于、大于、大于或等于
4	&&	逻辑与
3	||	逻辑或
2	=	赋值符(包括复合赋值符)
1	,	逗号

如果两个运算符的优先级相同，则按照从左往右的顺序进行计算。如果记不清两个运算符的优先级大小，则可以使用小括号()。

举例：

```
func main() {
    var a bool
    a = 2 == 1+1                    //等价于 a = (2 == (1+1))
    fmt.Println(a)                  //输出 true
}
```

有时使用小括号不是为了改变运算顺序，而是为了提高代码的可读性，这是一种很好的编程实践，减轻了程序员记忆运算符优先级的负担。

```
(a < 10) && (b > 20)                //推荐的做法，尽管小括号是多余的
a < 10 && b > 20                    //不推荐
```

3.7　小结

本章讲述了各种运算符的用法，包括算术运算符、关系运算符、逻辑运算符、位运算符等。将赋值符“＝”与其他运算符相结合就构成了复合赋值运算符，如“＋＝”。求解一个表达式的值时，必须按照运算符的优先级从高到低的顺序进行计算(括号除外)。另外，逻辑运算符还有一个短路求值特性，利用这一特性可以编写出更简洁、更高效的代码。

练习题

1. 编程计算下列各表达式的值：

(1) 5 ＋ 6

(2) 5 － 6

(3) 5 * 6

(4) 10 / 4

(5) 10 % 4

2. 写出表达式 a ＋ b － 5 的运算符和操作数。

3. 执行下列代码，其输出结果是_____。

```
fmt.Println(100 - 25 * 3 % 4)
```

4. 已知 x ＝ 3，执行 x *＝ 2 ＋ 1 语句后 x 的值等于_____。

5. 如果数据在计算机内存中用 1 字节存储，则－5 的补码为_____。

6. 代码 5 ＞ 3＋2 的执行结果是_____。

7. 代码 fmt.Println("a" ＜ "b")的执行结果是_____。

8. 代码 fmt.Println(124 ＋ 3.0) 的执行结果是_____。

9. 请举例说明短路求值特性。

10. 已知 x ＝ true，y ＝ false，求下列各表达式的值：

(1) x || y　　(2) x && y　　(3) !x

11. 已知 a = 6,b = 5,求下列各表达式的值:

(1) a & b　　(2) a | b　　(3) a ^ b

(4) ^a　　(5) a >> 1　　(6) b << 1

12. 写出下列代码的输出结果:

```
func main() {
    a, b := 5, 2
    c := (a != 0 && (b/a) > 0)
    fmt.Println(c)
}
```

13. 编写程序,输出一个三位数 254 的个位数字、十位数字和百位数字。

14. 阅读下列代码,写出程序的输出结果:

```
func main() {
    var x, y uint8 = 5, 3
    fmt.Println(x & y)
    fmt.Println(x | y)
    fmt.Println(^x)
    fmt.Println(x ^ y)
    fmt.Println(x &^ y)
    fmt.Println(x << 1)
    fmt.Println(x >> 1)
}
```