> 慕课网慕课教程 3. 运算符和表达式涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

1. 算数运算符

2. 关系运算符

== != > < >= <=

3. 逻辑运算符

&&	**所谓逻辑与运算符。如果两个操作数都非零，则条件变为真**
||	所谓的逻辑或操作。如果任何两个操作数是非零，则条件变为真
!	所谓逻辑非运算符。使用反转操作数的逻辑状态。如果条件为真，那么逻辑非操后结果为假

这个和 python 不一样，python 中使用 and or 来连接

```
package main

import "fmt"

func main() {
   var a bool = true
   var b bool = false
   if ( a && b ) {
      fmt.Printf("第一行 - 条件为 true\n" )
   }
   if ( a || b ) {
      fmt.Printf("第二行 - 条件为 true\n" )
   }

   a = false
   b = true
   if ( a && b ) {
      fmt.Printf("第三行 - 条件为 true\n" )
   } else {
      fmt.Printf("第三行 - 条件为 false\n" )
   }
   if ( !(a && b) ) {
      fmt.Printf("第四行 - 条件为 true\n" )
   }
}
```

4. 位运算符

位运算符对整数在内存中的二进制位进行操作。
下表列出了位运算符 &, |, 和 ^ 的计算：

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

假定 A = 60; B = 13; 其二进制数转换为：
A = 0011 1100
B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001
Go 语言支持的位运算符如下表所示。假定 A 为 60，B 为 13：

运算符	描述	实例
&	按位与运算符 "&" 是双目运算符。 其功能是参与运算的两数各对应的二进位相与。	(A & B) 结果为 12, 二进制为 0000 1100
|	按位或运算符 "|" 是双目运算符。 其功能是参与运算的两数各对应的二进位相或	(A | B) 结果为 61, 二进制为 0011 1101
^	按位异或运算符 "^" 是双目运算符。 其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为 1。	(A ^ B) 结果为 49, 二进制为 0011 0001
<<	左移运算符 "<<“是双目运算符。左移 n 位就是乘以 2 的 n 次方。 其功能把”<<“左边的运算数的各二进位全部左移若干位，由”<<" 右边的数指定移动的位数，高位丢弃，低位补 0。	A << 2 结果为 240 ，二进制为 1111 0000
>>	右移运算符 ">>" 是双目运算符。右移 n 位就是除以 2 的 n 次方。	
其功能是把 ">>“左边的运算数的各二进位全部右移若干位，”>>" 右边的数指定移动的位数。	A >> 2 结果为 15 ，二进制为 0000 1111	

```
package main

import "fmt"

func main() {

   var a uint = 60
   var b uint = 13
   var c uint = 0

   c = a & b
   fmt.Printf("第一行 - c 的值为 %d\n", c )

   c = a | b
   fmt.Printf("第二行 - c 的值为 %d\n", c )

   c = a ^ b
   fmt.Printf("第三行 - c 的值为 %d\n", c )
```

```
   c = a << 2
   fmt.Printf("第四行 - c 的值为 %d\n", c )

   c = a >> 2
   fmt.Printf("第五行 - c 的值为 %d\n", c )
}
```

赋值运算符

下表列出了所有 Go 语言的赋值运算符。

运算符	描述	实例
=	简单的赋值运算符，将一个表达式的值赋给一个左值	C = A + B 将 A + B 表达式结果赋值给 C
+=	相加后再赋值	C += A 等于 C = C + A
-=	相减后再赋值	C -= A 等于 C = C - A
*=	相乘后再赋值	C *= A 等于 C = C * A
/=	相除后再赋值	C /= A 等于 C = C / A
%=	求余后再赋值	C %= A 等于 C = C % A
<<=	左移后赋值	C <<= 2 等于 C = C << 2
>>=	右移后赋值	C >>= 2 等于 C = C >> 2
&=	按位与后赋值	C &= 2 等于 C = C & 2
^=	按位异或后赋值	C ^= 2 等于 C = C ^ 2
|=	按位或后赋值	C |= 2 等于 C = C | 2

```
package main

import "fmt"

func main() {
   var a int = 21
   var c int

   c =  a
   fmt.Printf("第 1 行 - =  运算符实例，c 值为 = %d\n", c )

   c +=  a
   fmt.Printf("第 2 行 - += 运算符实例，c 值为 = %d\n", c )

   c -=  a
```

```
   fmt.Printf("第 3 行 - -= 运算符实例，c 值为 = %d\n", c )

   c *=  a
   fmt.Printf("第 4 行 - *= 运算符实例，c 值为 = %d\n", c )

   c /=  a
   fmt.Printf("第 5 行 - /= 运算符实例，c 值为 = %d\n", c )

   c  = 200;

   c <<=  2
   fmt.Printf("第 6行  - <<= 运算符实例，c 值为 = %d\n", c )

   c >>=  2
   fmt.Printf("第 7 行 - >>= 运算符实例，c 值为 = %d\n", c )

   c &=  2
   fmt.Printf("第 8 行 - &= 运算符实例，c 值为 = %d\n", c )

   c ^=  2
   fmt.Printf("第 9 行 - ^= 运算符实例，c 值为 = %d\n", c )

   c |=  2
   fmt.Printf("第 10 行 - |= 运算符实例，c 值为 = %d\n", c )

}
```

其他运算符

此处讲解一下什么是地址

运算符	描述	实例
&	返回变量存储地址	&a; 将给出变量的实际地址。
*	指针变量。	*a; 是一个指针变量

```
package main

import "fmt"

func main() {
   var a int = 4
   var b int32
   var c float32
   var ptr *int
```

```
    fmt.Printf("第 1 行 - a 变量类型为 = %T\n", a );
    fmt.Printf("第 2 行 - b 变量类型为 = %T\n", b );
    fmt.Printf("第 3 行 - c 变量类型为 = %T\n", c );


    ptr = &a
    fmt.Printf("a 的值为  %d\n", a);
    fmt.Printf("*ptr 为 %d\n", *ptr);
}
```

运算符优先级

有些运算符拥有较高的优先级，二元运算符的运算方向均是从左至右。下表列出了所有运算符以及它们的优先级，由上至下代表优先级由高到低

优先级	分类	运算符	结合性
1	逗号运算符	,	从左到右
2	赋值运算符	=、+=、-=、*=、/=、 %=、 >=、 <<=、&=、^=、|=	从右到左
3	逻辑或	||	从左到右
4	逻辑与	&&	从左到右
5	按位或	|	从左到右
6	按位异或	^	从左到右
7	按位与	&	从左到右
8	相等 / 不等	==、!=	从左到右
9	关系运算符	<、<=、>、>=	从左到右
10	位移运算符	<<、>>	从左到右
11	加法 / 减法	+、-	从左到右
12	乘法 / 除法 / 取余	*（乘号）、/、%	从左到右
13	单目运算符	!、*（指针）、& 、++、–、+（正号）、-（负号）	从右到左
14	后缀运算符	()、[]、->	从左到右

当然，你可以通过使用括号来临时提升某个表达式的整体运算优先级。

```
package main

import "fmt"
```

```
func main() {
   var a int = 20
   var b int = 10
   var c int = 15
   var d int = 5
   var e int;

   e = (a + b) * c / d;
   fmt.Printf("(a + b) * c / d 的值为 : %d\n",  e );

   e = ((a + b) * c) / d;
   fmt.Printf("((a + b) * c) / d 的值为  : %d\n" ,  e );

   e = (a + b) * (c / d);
   fmt.Printf("(a + b) * (c / d) 的值为  : %d\n",  e );

   e = a + (b * c) / d;
   fmt.Printf("a + (b * c) / d 的值为  : %d\n" ,  e );
}
```