

Go 语言运算符

运算符用于在程序运行时执行数学或逻辑运算。

Go 语言内置的运算符有：

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符
- 其他运算符

接下来让我们来详细看看各个运算符的介绍。

算术运算符

下表列出了所有Go语言的算术运算符。假定 A 值为 10，B 值为 20。

运算符	描述	实例
+	相加	A + B 输出结果 30
-	相减	A - B 输出结果 -10
*	相乘	A * B 输出结果 200
/	相除	B / A 输出结果 2
%	求余	B % A 输出结果 0
++	自增	A++ 输出结果 11
--	自减	A-- 输出结果 9

以下实例演示了各个算术运算符的用法：

```
package main

import "fmt"

func main() {

    var a int = 21
```

```
var b int = 10
var c int

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 / b
fmt.Printf("第四行 - c 的值为 %d\n", c )
c = a % b
fmt.Printf("第五行 - c 的值为 %d\n", c )
a++
fmt.Printf("第六行 - a 的值为 %d\n", a )
a=21 // 为了方便测试, a 这里重新赋值为 21
a--
fmt.Printf("第七行 - a 的值为 %d\n", a )
}
```

以上实例运行结果：

```
第一行 - c 的值为 31
第二行 - c 的值为 11
第三行 - c 的值为 210
第四行 - c 的值为 2
第五行 - c 的值为 1
第六行 - a 的值为 22
第七行 - a 的值为 20
```

关系运算符

下表列出了所有Go语言的关系运算符。假定 A 值为 10，B 值为 20。

运算符	描述	实例
==	检查两个值是否相等，如果相等返回 True 否则返回 False。	(A == B) 为 False
!=	检查两个值是否不相等，如果不相等返回 True 否则返回 False。	(A != B) 为 True
>	检查左边值是否大于右边值，如果是返回 True 否则返回 False。	(A > B) 为 False
<	检查左边值是否小于右边值，如果是返回 True 否则返回 False。	(A < B) 为 True
>=	检查左边值是否大于等于右边值，如果是返回 True 否则返回 False。	(A >= B) 为 False

<=	检查左边值是否小于等于右边值，如果是返回 True 否则返回 False。	(A <= B) 为 True
----	---------------------------------------	-----------------

以下实例演示了关系运算符的用法：

```
package main

import "fmt"

func main() {
    var a int = 21
    var b int = 10

    if( a == b ) {
        fmt.Printf("第一行 - a 等于 b\n" )
    } else {
        fmt.Printf("第一行 - a 不等于 b\n" )
    }

    if ( a < b ) {
        fmt.Printf("第二行 - a 小于 b\n" )
    } else {
        fmt.Printf("第二行 - a 不小于 b\n" )
    }

    if ( a > b ) {
        fmt.Printf("第三行 - a 大于 b\n" )
    } else {
        fmt.Printf("第三行 - a 不大于 b\n" )
    }

    /* Lets change value of a and b */
    a = 5
    b = 20
    if ( a <= b ) {
        fmt.Printf("第四行 - a 小于等于 b\n" )
    }
    if ( b >= a ) {
        fmt.Printf("第五行 - b 大于等于 a\n" )
    }
}
```

以上实例运行结果：

```
第一行 - a 不等于 b
第二行 - a 不小于 b
第三行 - a 大于 b
第四行 - a 小于等于 b
第五行 - b 大于等于 a
```

逻辑运算符

下表列出了所有Go语言的逻辑运算符。假定 A 值为 True , B 值为 False。

运算符	描述	实例
&&	逻辑 AND 运算符。 如果两边的操作数都是 True , 则条件 True , 否则为 False。	(A && B) 为 False
	逻辑 OR 运算符。 如果两边的操作数有一个 True , 则条件 True , 否则为 False。	(A B) 为 True
!	逻辑 NOT 运算符。 如果条件为 True , 则逻辑 NOT 条件 False , 否则为 True。	!(A && B) 为 True

以下实例演示了逻辑运算符的用法：

```
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" )
    }
    /* 修改 a 和 b 的值 */
    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" )
    }
}
```

以上实例运行结果：

第二行 - 条件为 true

第三行 - 条件为 false

第四行 - 条件为 true

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	A << 2 结果为 240 ，二进制为 1111 0000

	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。	
>>	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

以下实例演示了位运算符的用法：

```
package main

import "fmt"

func main() {

    var a uint = 60    /* 60 = 0011 1100 */
    var b uint = 13    /* 13 = 0000 1101 */
    var c uint = 0

    c = a & b          /* 12 = 0000 1100 */
    fmt.Printf("第一行 - c 的值为 %d\n", c )

    c = a | b          /* 61 = 0011 1101 */
    fmt.Printf("第二行 - c 的值为 %d\n", c )

    c = a ^ b          /* 49 = 0011 0001 */
    fmt.Printf("第三行 - c 的值为 %d\n", c )

    c = a << 2          /* 240 = 1111 0000 */
    fmt.Printf("第四行 - c 的值为 %d\n", c )

    c = a >> 2          /* 15 = 0000 1111 */
    fmt.Printf("第五行 - c 的值为 %d\n", c )
}
```

以上实例运行结果：

```
第一行 - c 的值为 12
第二行 - c 的值为 61
第三行 - c 的值为 49
第四行 - c 的值为 240
第五行 - c 的值为 15
```

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

以下实例演示了赋值运算符的用法：

```
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 )

}
```

以上实例运行结果：

```
第 1 行 - = 运算符实例, c 值为 = 21
第 2 行 - += 运算符实例, c 值为 = 42
第 3 行 - -= 运算符实例, c 值为 = 21
第 4 行 - *= 运算符实例, c 值为 = 441
第 5 行 - /= 运算符实例, c 值为 = 21
第 6行  - <<= 运算符实例, c 值为 = 800
第 7 行 - >>= 运算符实例, c 值为 = 200
第 8 行 - &= 运算符实例, c 值为 = 0
第 9 行 - ^= 运算符实例, c 值为 = 2
第 10 行 - |= 运算符实例, c 值为 = 2
```

其他运算符

下表列出了Go语言的其他运算符。

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

以下实例演示了其他运算符的用法：


```
package main

import "fmt"

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

    /* 运算符实例 */
    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 );

    /* & 和 * 运算符实例 */
    ptr = &a    /* 'ptr' 包含了 'a' 变量的地址 */
    fmt.Printf("a 的值为  %d\n", a);
    fmt.Printf("*ptr 为  %d\n", *ptr);
}
```

以上实例运行结果：

```
第 1 行 - a 变量类型为 = int
第 2 行 - b 变量类型为 = int32
第 3 行 - c 变量类型为 = float32
a 的值为  4
*ptr 为 4
```

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	运算符
7	^ !
6	* / % << >> & ^
5	+ - ^
4	== != < <= >= >
3	<-
2	&&

1	
---	--

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。

以上实例运行结果：

```
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;      // ( 30 * 15 ) / 5
    fmt.Printf("(a + b) * c / d 的值为 : %d\n", e );

    e = ((a + b) * c) / d;    // (30 * 15 ) / 5
    fmt.Printf("((a + b) * c) / d 的值为 : %d\n" , e );

    e = (a + b) * (c / d);    // (30) * (15/5)
    fmt.Printf("(a + b) * (c / d) 的值为 : %d\n", e );


    e = a + (b * c) / d;      // 20 + (150/5)
    fmt.Printf("a + (b * c) / d 的值为 : %d\n" , e );
}
```

以上实例运行结果：


```
(a + b) * c / d 的值为 : 90
((a + b) * c) / d 的值为 : 90
(a + b) * (c / d) 的值为 : 90
a + (b * c) / d 的值为 : 50
```

← Go 语言常量

Go 语言条件语句 →



1 篇笔记

 写笔记



指针变量 * 和地址值 & 的区别：指针变量保存的是一个地址值，会分配独立的内存来存储一个整型数字。当变量前面有 * 标识时，才等同于 & 的用法，否则会直接输出一个整型数字。

```
func main() {  
    var a int = 4  
    var ptr *int  
    ptr = &a  
    println("a的值为", a);    // 4  
    println("*ptr为", *ptr);  // 4  
    println("ptr为", ptr);    // 824633794744  
}
```

Gidot 3个月前 (12-20)