Go 位操作技巧

鸟窝出品

1 基本操作

```
// AND 与
0b0001 & 0b0010 = 0000

// OR 或
0b0001 | 0b0010 = 0011

// XOR 异或
0b0001 ^ 0b0010 = 0011

// NOT (for int8) 取反，类似其他语言的 ~ 操作
^0b0010 = 11111101

// Bitclear (AND NOT), 清除指定位
0b0011 &^ 0b0010 = 0001

// Left Shift, 左移
1 << 2 = 4  // 1 * 2^2

// Right Shift, 右移
1 >> 2 = 0  // 1 / 2^2
```

2 bits 包中的位操作函数

```
import "math/bits"

// Count ones, 统计 1 的个数
bits.OnesCount8(0b00101110) = 4

// Count significant bits, 统计有效位数
bits.Len8(0b00101110) = 6
bits.Len8(0b00000000) = 0

// Count leading zeros, 统计前导 0 的个数
bits.LeadingZeros8(0b00101110) = 2
bits.LeadingZeros8(0b00000000) = 8

// Count trailing zeros, 统计末尾 0 的个数
bits.TrailingZeros8(0b00101110) = 1
bits.TrailingZeros8(0b00000000) = 8

// Rotate left, 左旋，旋转 n 位
bits.RotateLeft8(0b00101110, 3) = 01110001

// Rotate right, 右旋，旋转 n 位
bits.RotateLeft8(0b00101110, -3) = 11000101

// Reverse bits, 反转位，末位变首位
bits.Reverse8(0b00101110) = 01110100

// Reverse bytes, 反转字节，末字节变首字节
bits.ReverseBytes16(0x00ef) = 0xef00
```

3 整数算术运算

```
// Multiply by 2^n, 乘以 2 的 n 次方
x = y << n // 1<<8 = 256

// Divide by 2^n, 除以 2 的 n 次方
x = y >> n // 512>>8 = 2

// Check if x is even, 检查 x 是否为偶数
(x & 1) == 0 // 512&1 = 0

// Check if x is a power of 2, 检查 x 是否为 2 的幂
x != 0 && x&(x-1) == 0 // 512&(512-1)= 0

// Check if a number is divisible by 8, 检查一个数是否能被 8 整除
((n >> 3) << 3) == n // 512>>3<<3 = 512

// Check if x and y have opposite signs, 检查 x 和 y 是否异号
(x ^ y) < 0 // 512^-1024= -512
```

4 单 bit 变换

```
// Set the nth bit, 设置第 n 位为 1, n 从 0 从右开始
y = x | (1 << n) // 10000000 | (1<<3) = 10001000

// Unset the nth bit, 设置第 n 位为 0
y = x &^ (1 << n) // 01111111 &^ (1<<3) = 01110111

// Toggle the nth bit, 反转第 n 位
y = x ^ (1 << n) // 10000000 ^ (1<<3) = 10001000

// Toggle all bits except nth bit, 反转除第 n 位外的所有位
y = ^(x ^ (1 << n)) // ^(10000000 ^ (1<<3)) = 01110111

// Toggle rightmost m bits, 反转最右边的 m 位
y = x ^ ^(-1 << n) // 10000000 ^^ (-1<<3) = 10000111

// Test if the nth bit is set, 测试第 n 位是否为 1
(x & (1 << n)) != 0 // 10000000 & (1<<3) = 00000000

// Turn off rightmost 1-bit, 将最右边的 1 位设置为 0, 其余位不变
y = x & (x - 1) // 01111111 & (128-1) = 01111110

// Isolate rightmost 1-bit, 保留最右边的 1 位, 其余位设置为 0
y = x & (-x) // 01111000 & (-120) = 00001000

// Right propagate rightmost 1-bit, 将最右边的 1 位的右边所有位设置为 1
y = x | (x - 1) // 01011000 | (88-1) = 01011111

// Turn on rightmost 0-bit // 将最右边的 0 位设置为 1, 其余位不变
y = x | (x + 1) // 01011000 | (88+1) = 01011001

// Isolate rightmost 0-bit, 保留最右边的 0 位, 其余位设置为 1
y = ^x & (x + 1) // ^00001001 & (88+1) = 00000010
```

关注公众号