理解位运算及使用场景

作者：kuuyee
原文链接：https://ld246.com/article/1445898392096
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

最近在看APUE，函数**umask**的例子用到了位运算，认为这是个非常适合使用位运算的场景，有必要记一下。例子代码基于golang,因为最近在学习golang.

位运算

先来看下位运算的定义：**程序中的所有数在计算机内存中都是以二进制的形式储存的。位运算说穿了就是直接对整数在内存中的二进制位进行操作。**摘自百度百科

比如，**&运算本来是一个逻辑运算符，但整数与整数之间也可以进行&算。举个例子，6的二进制是110，11的二进制是1011，那么6 & 11的结果就是2，它是二进制对应位进行逻辑运算的结果（0示False ，1表示True**，空位都当**0**处理）,下面的代码是用go实现的：

```
func base() {
    a := 6        //    0110
    b := 11     //    1011
                //    ----
                // &  0010 => 2
                // |  1111 => 15
                // ^  1101 => 13
                // &^ 0100 => 4
    fmt.Println(a & b)
    fmt.Println(a | b)
    fmt.Println(a ^ b)
    fmt.Println(a &^ b)
}
```

四个位运算符说明如下：

```
0110 & 1011    = 0010 AND 都为1。
0110 | 1011 = 1111 OR 至少一个为1。
0110 ^ 1011 = 1101 XOR 只能一个为1。
0110 &^ 1011 = 0100 AND NOT 清除标志位。
```

应用场景-Umask

只是位运算的话理解起来挺容易的，但是这种位运算有毛用呢？适于用什么场景？下面我用Unix系统的**mask**概念来实践下位运算。关于umask的概念请参阅http://linux.vbird.org/linux_basic/0220filemanager.php#umask。简单来讲，Unix系统对于文件的权限用9个权限位来控制：

```
[-][rwx][r-x][r--]
 1  234  567  890
```

- **r**:可读 4
- **w**:可写 2
- **x**:可执行 1
- **-**:表示此权限被去除

第一位是用来表示是文件还是目录，先不用管它，主要是后面9位。我们经常在授权是用的到**644**,**755**

是用**r,w.x**这三个值相加得出的。为什么值分别是**4,2,1**呢，我们把go语言**sys**包中的源码拿出来看看明白了：

```
const (
    S_IRUSR = 0x100 //用户可读
    S_IWUSR = 0x80    //用户可写
    S_IXUSR = 0x40    //用户可执行
    S_IRGRP = 0x20    //组可读
    S_IWGRP = 0x10    //组可写
    S_IXGRP = 0x8    //组可执行
    S_IROTH = 0x4    //其它可读
    S_IWOTH = 0x2    //其它可写
    S_IXOTH = 0x1    //其它可执行
)
```

这是sys包中定义的一些常量,我们来打印下这些都是啥玩意

```
    fmt.Printf("%9b %3d %s\n", S_IRUSR, S_IRUSR, "用户读")
    fmt.Printf("%9b %3d %s\n", S_IWUSR, S_IWUSR, "用户写")
    fmt.Printf("%9b %3d %s\n", S_IXUSR, S_IXUSR, "用户执行")

    fmt.Printf("%9b %3d %s\n", S_IRGRP, S_IRGRP, "组读 *")
    fmt.Printf("%9b %3d %s\n", S_IWGRP, S_IWGRP, "组写 *")
    fmt.Printf("%9b %3d %s\n", S_IXGRP, S_IXGRP, "组执行")

    fmt.Printf("%9b %3d %s\n", S_IROTH, S_IROTH, "其它读 *")
    fmt.Printf("%9b %3d %s\n", S_IWOTH, S_IWOTH, "其它写 *")
    fmt.Printf("%9b %3d %s\n", S_IXOTH, S_IXOTH, "其它执行")

// 输出
100000000 256 用户可读
 10000000 128 用户可写
  1000000  64 用户可执行
   100000  32 组可读
    10000  16 组可写
     1000   8 组可执行
      100   4 其它可读
       10   2 其它可写
        1   1 其它可执行
```

看明白了吧，其实就是把九个权限位置分别标志为**1**，用二进制可以很清楚的表示权限位，**4,2,1**也就这么来的。那么**umask**的就可以利用这个位运算，代码如下：

```
package main

import (
    "fmt"
    "golang.org/x/sys/unix"
    "os"
)

func main() {
```

```
    unix.Umask(0)
    _, err := os.Create("foo")
    if err != nil {
        fmt.Println("Create Error")
    }
    unix.Umask(unix.S_IRGRP | unix.S_IWGRP | unix.S_IROTH | unix.S_IWOTH)
    _, err2 := os.Create("bar")

    if err2 != nil {
        fmt.Println("Create Error")
    }
}
```

上面的代码可以看到，unix.S_IRGRP | unix.S_IWGRP | unix.S_IROTH | unix.S_IWOTH就利用了位算，程序先后创建了两个文件**foo**，**bar**, 创建**bar**文件时，文件初始权限把**组**和**其它**中的**可读**和**可写**除了，所以我们用ls -la foo bar命令可以看到输出如下：

```
[vagrant@mydev ~]$ ls -la foo bar
-rw-------. 1 vagrant vagrant 0 Oct 26 02:54 bar
-rw-rw-rw-. 1 vagrant vagrant 0 Oct 26 02:54 foo
```

再用9个权限位的二进制图说明下：

```
100000000 256 用户可读
 10000000 128 用户可写
  1000000  64 用户可执行
   100000  32 组可读 *
    10000  16 组可写 *
     1000   8 组可执行
      100   4 其它可读 *
       10   2 其它可写 *
        1   1 其它可执行
---------
   110110    unix.S_IRGRP | unix.S_IWGRP | unix.S_IROTH | unix.S_IWOTH
```

umask四个参数我用型号标出来了，那经过**|**位运算最终结果就是**110110**,言外之意就是**umask**标志为**1**的权限位去除了。

最后我把权限位二进制打印代码贴出来

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/53393b50ab661d2194341f6a67dba10.go?embed=true" width="99%" height="600"></iframe>

原文链接：理解位运算及使用场景