浙江科技学院学报,第 15 卷第 4 期,2003 年 12 月
Journal of Zhejiang University of Science and Technology
Vol.15 No.4, Dec. 2003

位运算标记对象的当前状态

王永芬
(杭州广播电视大学 基础部,浙江 杭州 310012)

摘 要:讨论了位运算对对象状态标记的应用,给出了对象双值状态和多值状态的设置与判别方法。在面向对象技术中,通过位运算可以实现对数据冗余的封装,它的应用不但保证了数据的一致性,而且有利于系统的维护与扩充。

关键词:位运算;位标记;数据冗余
中图分类号: TP312 **文献标识码**: A **文章编号**: 1671 - 8798(2003)04 - 0228 - 04

位运算属于机器级指令,只适应整数类型的数据运算。C/C ++ 语言提供单目求反操作以及双目求和、或、异或、左移和右移等操作。用位运算的特点是计算速度快,例如判别一个整数 n 奇数只要判别 $n\&1$ 的真伪;2^n 是对 1 左移 n 位,即表达式 $1 \ll n$ 值;对整数类型变量乘 2^n 和除 2^n 的计算结果,都可以通过左移和右移 n 位来实现。

位运算的主要用途是标记对象的当前状态[1]。在实体的数据结构或对象类设计中,设置一个整数变量 Flag 标记对象的多维状态[2,3],既节省了系统运行时存储开销,又提高了系统的可维护性。本文用 C/C ++ 语言,讨论运用位运算实现对象状态的设置与判别,以及数据冗余处理等方面的应用技术。

1 双值状态标记

当标记的状态"真与假"(或"是与非")时,只用变量的一位标记即可,不需要专门设计一个变量,只一个标记变量就可以标记许多种状态。设有一个对象的标记状态成员变量为:

```
UINT nFlag; // UINT 是 unsigned int 的宏或类型定义
```

它有 32 个位(根据需要也可以设计成 16 位或 8 位的变量),如果完全用来标记"真与假"状态,可标记 32 种状态。当然,所有的状态不一定完全只是"真与假"。假如 nFlag 的最右边的 6 个位都表示的是"真与假"状态,首先用十六进制定义它们的状态值宏:

```
# define STATE_1      0x00000001      // 二进制中,第一位为 1,其他位为 0
# define STATE_2      0x00000002      // 二进制中,第二位为 1,其他位为 0
# define STATE_3      0x00000004      // 二进制中,第三位为 1,其他位为 0
# define STATE_4      0x00000008      // 二进制中,第四位为 1,其他位为 0
# define STATE_5      0x00000010      // 二进制中,第五位为 1,其他位为 0
# define STATE_6      0x00000020      // 二进制中,第六位为 1,其他位为 0
```

十六进制中的每一位代表四位二进制,定义是非位标记常数按 1,2,4,8 的规律前进的。对于一种状态直,开发人员很难在编程过程中记住,所以采用方便记忆的宏常数,例如打印机是否已经打开,用宏常数 PRINTER_OPEN。注意,以上常数定义也可以采用枚举来定义(见后面应用实例)。

收稿日期: 2003 - 03 - 12

作者简介: 王永芬(1968—),女,浙江上虞人,讲师,主要从事工程力学、流体力学的教学与科研工作。

把某位设置成真:如果把变量 nFlag 的某一位,例如以上例子的第三位,设置成“真”,则可以用以下语句实现:nFlag | = STATE_3;执行上述语句后,第三位变成了 1,而其他位没有改变。

把某位设置成假:同样地,希望把以上例子的第三位设置成非,则可以用以下语句实现:nFlag &= (~STATE_3);执行上述语句后,第三位变成了 0,而其他位没有改变。

判别某位的是与非:要判别某位的是与非,例如希望判别以上例子的第三位的是与非,只要判别表达式 nFlag&STATE3 是否为 0 即可。

2　多值状态标记

当需要表达的状态超过两种值时,需要更多的位来表示,设某状态有 K 种情况需要表示,则需要的位数是满足 $2^n \geqslant K$ 条件的最小的 n 值。继续采用上面的对象标记变量 nFlag,从第 7 位到第 9 位(3 个位)表达 6 种状态。首先定义各种状态值的宏:

```
# define MUL_STATE_1        0x00000000          // 三位都是 0 表示状态 1
# define MUL_STATE_2        0x00000040          // 第三位是 1 表示状态 2
# define MUL_STATE_3        0x00000080          // 第四位是 1 表示状态 3
# define MUL_STATE_4        (STATE_2|STATE_3)   // 第三四都位是 1 表示状态 4
# define MUL_STATE_5        0x00000100          // 第五位是 1 表示状态 5
# define MUL_STATE_6        (STATE_4|STATE_5)   // 第三五位都是 1 表示状态 6
# define FULL_STATE         (STATE_2| STATE_3| STATE_5)  // 过滤器
```

在以上的常数定义中,如果光考虑三位得值,而不去考虑其他得位,则 6 种状态在三位上的数据值分别是 0,1,2,3,4,5。最后一个宏 FULL_STATE 是一个满位常数,它起着状态的过滤作用,是多状态位运算中必需的。

设置成某一种状态:把变量 nFlag 的 7~9 位设置成某一种状态 state(state 是以上情况 STATE_1~STATE_6 之一),则可以用下列语句实现:

```
state &= FULL_STATE;          // 过滤语句——过滤掉超出 7~9 位的值
nFlag &= (~FULL_STATE);       // 把 7~9 位全部设置成 0
nFlag |= state;               // 设置成状态 state
```

以上的过滤语句,是为了过滤掉超出 7~9 位以外的数据侵略。

状态判别:获得状态可用以下表达式即可得 nFlag&FULL_STATE

3　数据冗余的封装

数据冗余设置是通过牺牲空间而获得计算速度的一种办法。当一个系统的数据结构或数据库中有数据冗余存在时,必须加上一些特殊的处理以保证模型中的数据一致性。用位标记法是一种很好解决问题的办法,特别是与面向对象技术中的封装性相结合,能够获得更好的数据一致性。

问题提法:设在一对象中有一因变量 y,它不是对象的独立变量,需要通过比较复杂的计算获得结果,并且又被自己的其他过程和其他对象的过程频繁调用。为了保证计算速度,特设置一对象成员变量 m_y 以存放 y 的计算结果,很显然 m_y 属于数据冗余。当 y 已被计算,其他过程调用 m_y 以获得 y 的值,否则调用 y 的计算函数。当 y 值被重新计算时,需要对 m_y 的值作出修改。

要达到不因为数据冗余的存在而影响对象的派生发展和对象对外消息的传递[4],用位标记和对象的封装技术对变量 y 进行如下处理:

(1) 用标记变量中的一位标记函数 $y()$ 是否被计算,如果是则记录计算结果,并打上位标记;

(2) 把 $y()$ 封装成保护性成员函数,可以被继承与派生,但是不允许其他家族的对象调用;

(3) 把 m_y 封装成私有的成员变量,不允许被继承对象和其他家族对象调用;

(4) 设置一个对外透明的非多态函数 $Y()$,在 $Y()$ 中进行逻辑判别和位设置和中间结果保存。

当对象中的 y 函数需要派生时,必须从成员函数 $y()$ 中发展,其他过程只能调用 $Y()$,并且 $Y()$ 不允许被派生(也不需要派生)。如此处理,不但彻底封装了数据冗余的存在,而且后续的开发人员(对象类的派生开发与使用 y 消息的开发)都不感觉到有数据冗余的存在,同时也保证了系统的计算速度。

4 对象的可扩充性

在软件开发过程中,常常遇到对象的状态标记需要扩充的情况,既在原来系统的基础上增加新的状态。特别是面向对象技术开发的系统,在任何时期都无法判别系统将有什么对象被派生,更无法判别派生的对象有什么状态需要在数据中标记。如果在对象类设计中每种状态都用一个变量表达,那么当系统的对象状态需要扩充时,也只能增加新的变量来满足要求。增加变量的结果不但增加了空间的开销,更严重的是由于数据结构格式的改变,很难满足系统的数据兼容性,系统的维护性被破坏。反之,如果用固定数量(一般是一个)位标志器变量表达对象的状态,则由于标志器中往往存在空余的空间位,空间位可以作为新状态的表达空间,既满足了新功能的增加,又满足了数据的向上兼容。

5 应用实例

用 C++ 语言设计一个《锅炉热力计算系统》的用户参数类,在此类中,有四种状态需要标记,它们分别是:①锅炉的类型,分室燃炉、层燃炉、硫化床等,需要有扩充能力,最常用的是室燃炉;②锅炉出口蒸汽数,简称压力数,分单压、双压、三压和四压,最常用的是单压;③锅炉工作状态,分蒸汽锅炉和热水锅炉,常用的是蒸汽锅炉;④锅炉的燃料用量通过计算获得,并被以后的过程反复调用,所以需要标记是否已经计算。很显然,①与②状态超过两种情况需要多位表达,③与④状态都只有两种情况,各需要一位,而④状态是对数据冗余的处理。为了表达方便与将来扩充需要,①与②状态都用四位表示状态值,用四位表达多种情况的状态,在十六进制中可直接用 0,1,2,……表达。另外,②状态属于用变量的部分位表达一种整数状态。锅炉用户类定义如下(只阐述位运算部分):

(1) 类名称: CBoilerUserData

(2) 位标记器以及引用函数

```
private: UINT m_nFlag;
public: UINT &Flag() { return m_nFlag; }
```

(3) 一位状态标记运算函数

```
设置函数:void SetFlag(UINT nFlag) { Flag() |= nFlag;}
判别函数:bool JudgeFlag(UINT nFlag) { if(Flag()&nFlag) return true; else return false; }
取消函数:void BanFlag(UINT nFlag) { Flag() &= (~nFlag);}
```

(4) 位标记常数定义

(a)锅炉类型,用右边 4 位

```
MEIFEN_BOILER      = 0x00000000,    // 室燃炉
LIANTIAO_BOILER    = 0x00000001,    // 层燃炉
LIUHUA_BOILER      = 0x00000002,    // 硫化床
OTHER_BOILER       = 0x0000000f,    // 其他炉,前面可以扩充
FULL_BOILER        = 0x0000000f,    // 炉型过滤器
```

(b)锅炉压力数,用右边第二个 4 位

```
DANYA              = 0x00000000,    // 单压
SHUANGYA           = 0x00000010,    // 双压
SANYA              = 0x00000020,    // 三压
SIYA               = 0x00000030,    // 四压
```

FULL_ YASHU　　= 0x000000f0,　　//压力数过滤器

(c)工作类型,右边第 9 位

WT_ WATER　　= 0x00000100,　　// 是热水锅炉,否则是蒸汽锅炉

(d)已经求得燃料消耗量,右边第 10 位

HAVE_ CAL_ BP　　= 0x00000200　　// 标记已经求得燃料消耗量

(5)位标记器函数定义

(a)设置锅炉类型:void SetBoilerType(UINT nBoilerType) {

nBoilerType &= FULL_ BOILER;

Flag() &= (~FULL_ BOILER);

Flag() |= nBoilerType; }

(b)获得锅炉类型函数定义:UINT CHProjectData::BoilerType()

{ return Flag()&FULL_ BOILER; }

(c)用整数设置锅炉压力数:void SetDrumCounter(UINT n = 1){

int nFlag = FULL_ YASHU;

for(int i = 0; i < 32; i ++) if(! nFlag&1) nFlag>>= 1; else break;

SetDrumCounterFlag((n - 1)<<i); }

(d)得锅炉压力数:int DrumCounter() {

int nFlag = FULL_ YASHU;

for(int i = 0; i < 32; i ++) if(! nFlag&1) nFlag >>= 1; else

return (Flag()&FULL_ YASHU)>>i + 1; }

(e)设置成热水锅炉工作类型:void SetWaterBoiler(){ SetFlag(WT_ WATER); }

(f)设置成蒸汽锅炉工作类型:void SetSteamBoiler(){ BanFlag(WT_ WATER); }

(g)判别是否热水锅炉:bool IsWaterBoiler() { return JudgeFlag(WT_ WATER);}

(6)燃料量处理

(a)燃料量计算函数,可以被派生,但是不能够对外调用。

protected: virtual double Fbp() ;

(b)为了计算速度,对设置燃料量中间变量,为保证数据唯一性,对外彻底封装。

private: double m_ dBp;

(c)对外燃料量计算函数, 不能派生。

public: double FBp() {

if(! Flag()&HAVE_ CAL_ FBP) Fbp();

Flag() |= HAVE_ CAL_ FBP;

return m_ Bp; }

从以上类定义可以看出,通过位运算,用一个整数变量实现了对象多种状态的表达,而且只用了十位,其他位可留作系统扩充之用,十位中也有一些可扩充余地。

6 结　论

在系统软件与应用软件开发中都需要用位来标记对象的当前状态,位运算的熟练程度是判别软件开发水平的一个重要方面。熟练而充分地运用位运算可以编写出品质优秀的代码,从而提高系统的质量,特别是系统可维护性能的提高。对于不支持位运算的计算机语言可以通过整数算术运算来间接达到[5]。

(下转第 235 页)