一种引用计数机制的实现

作者：localvar
原文链接：https://ld246.com/article/1521526905388
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)

毫无疑问, 引用计数是一种非常有效的动态控制对象生命周期的机制. 我们最熟悉的引用计数实现可能要数COM的AddRef和Release了. 但这种机制也有明显的缺点, 那就是无法实现对对象死亡时间的精控制: 调用Release后, 就失去了对对象的控制, 虽然对象可能会被立即杀掉, 但我们无法保证这一点. 也程序的其他地方还对它拥有引用, 并且还会有一系列的AddRef和Release, 而只要计数不降到0, 对象一直活着, 甚至可能比你我更长寿.

为了更好的说明这一点, 请考虑下面的情况: 我们有某种类型的对象, 这种对象在程序运行过程中会不的被创建和杀死, 而所有活着的对象都被放在一个全局表格中. 由于表格拥有一个对象的引用, 所以表中不被程序其它部分使用的对象的计数将为1. 当程序要访问某个对象时, 就会通过一个键值从表格中到它, 递增其引用计数, 待访问完毕后, 再递减计数. 从以上可以看出, 我们要想杀掉一个对象, 只要去掉格对它的引用(也就是把引用计数减一)就可以了. 但这并不能确保对象被杀死, 因为程序的其它地方仍从表格中找到它, 并增加其计数; 更进一步, 我们可以在去掉表格的引用后, 把对象从表格中删除, 这样数就不会增加了, 但很不幸, 我们并不是在任何时候都能这样做, 有些时候没有"彻底死亡"的对象是不从表格中删除的.

那有没有两全其美的方法呢? 应该说还是有的. 引用计数通常用一个32位整数来表示, 它最大能支持几亿个引用, 但实践上, 能达到的最大值要远小于这个数字, 所以, 我们可以把其中的某些位挪作它用, 用表示对象是否已经被杀掉, 而不能再增加新的引用. 看下面的实现:

```
template<class T>
class CRefCount
{
private:
    // 使用第30位作为生存标志位
    static const LONG s_lAliveFlag = 0x40000000;
    volatile LONG m_lRef;

public:
    CRefCount() : m_lRef( s_lAliveFlag )
    {
    }

    bool AddRef()
    {
        LONG lRef;
        do{
            lRef = m_lRef;
            // 已经死亡了, 增加引用失败
            if( (lRef & s_lAliveFlag) == 0 )
                return false;
        } while( InterlockedCompareExchange(&m_lRef, lRef+1, lRef) != lRef );
        return true;
    }

    void Release()
    {
        if( InterlockedDecrement( &m_lRef ) == 0 )
        {
            T* pT = static_cast( this );
            delete pT;
        }
    }

    void Suicide()
```

```
    {
        // 注意: 调用此函数前应AddRef, 这样调用之后的Release才能正确删除对象
        InterlockedAnd( &m_lRef, ~s_lActiveFlag );
    }
};
```

程序很简单, 我就不做过多解释了, 但正像我在标题中写的, 它只是"一种引用计数的实现"方法而已, 和他实现相比, 它既有优点, 也有缺点, 所以使用时一定要根据实际情况, 选择最合适的方法. 另外, InterlockedAnd在VS2005中是编译器的一个intrinsic, 如果你使用的编译器不支持它, 可参考我的《对window互锁函数的补充》, 自己实现一个.