链滴

# 使用派生类对象通过成员函数指针调用基类虚函数之不可能性的证明

作者：localvar

原文链接：https://ld246.com/article/1521526125329

来源网站：链滴

希望大家没有被这么拗口的标题吓到:). 本文源于论坛上的这个问题:

```
struct base
{
    void foo()
    {
        cout << "base::foo" << endl;
    }
    virtual void bar()
    {
        cout << "base::bar" << endl;
    }
};
struct derived : base
{
    virtual void bar()
    {
        cout << "derived::bar" << endl;
    }
};
int __cdecl _tmain( int argc, _TCHAR* argv[] )
{
    void (base::*pfn)() = &base::bar;
    derived d;
    d.base::bar(); // 1
    (d.base::*pfn)(); // 2 想实现和上一行一样的输出, 但是编译失败
    return 0;
}
```

很明显, 标2的那一行是想使用派生类对象通过成员函数指针调用基类虚函数, 以实现与标1的那行相同输出, 但却无法编译通过. 这是个语法错误, 因为::运算符的优先级高于., 所以那一行会先计算base::*pfn, 然而*pfn并不是base的成员, 故有错误时很自然的. 那么是否可以通过修改那条语句来达到目的呢? 分了成员函数指针的实现后, 我发现, 至少在vc7.1和vc8上, 这是不可能的. 由于vc的标准兼容性已经非常,所以我怀疑c++标准就不支持这种调用, 但没有证实.

在上面的例子中, pfn指向的是虚函数bar, 但它也必须能指向普通成员函数foo. 当指向foo时, 它保存就是foo的入口地址; 然而当指向bar时, 直接保存这个地址就不行了,因为对base和derived来说, 这个址并不相同. vc对此的解决方法是由编译器加入了一系列的内部函数vcall. 一个类中的每个虚函数都有个唯一与之对应的vcall函数, 但在不同类之间, 这些vcall实际上是公用的. pfn指向的就是这些vcall中一个.

我们知道, 调用成员函数时要传递this指针. 一般情况下, 它是通过ecx寄存器传递的, 所以vcall的实现下所示:

```
    mov eax, dword ptr[ecx];
    jmp dword ptr [eax+xx]
```

第一句中, 由于ecx是this指针, 而一般vfptr是类的第一个成员, 所以它是把vfptr, 也就是vtable的地址到了eax中. 第二句里面的xx, 在32位计算机上, 是4的整数倍, 所以这一句的意思是: 跳转到vtable的第x/4项所指的地址上, 这个地址就是最终要调用的函数的入口.

明白了虚成员函数指针的实现, 就可以看那种调用为什么不能实现了. 让我们用反证法, 如果它能实现, 讨论方便就假设标2的那一句是正确的吧, 在进行调用时, 编译器首先要保证传递的是指向d的this指针, 然后还要保证this所指向的vfptr所指的是base 的vtable. 编译器能做到吗? 当然能, 它可以在调用前偷

修改vfptr使其指向base的vtable, 并在调用返回后再把它恢复过来, 但想想这样做在多线程环境中的果吧. 所以编译器是不可能这样做的, 也就是说"使用派生类对象通过成员函数指针调用基类虚函数是可能的".