COM 可重用性 组件 软件重用 软件工程 (17)

计算机科学 1999Vol. 26№. 11

45-48

COM的可重用性及其存在的问题

COM's Reuse and a Flaw in It

伦卿卿 杨宗源

(华东师范大学计算机科学系 上海 200062)

TP311.5

Abstract Considering the software reusable character, the component has highlighted the computer-science area as its prosperity of concept and industrialization. The mechanisms of reusing and extensibility in Microsoft OLE/COM specification are "containment" and "aggregation". The aggregation is specially remarkable except that it has made a flaw in the reuse of an existed class. Thus we provide an object-centered solution based on interface and IDs against the interface-centered solution of COM.

Keywords Component, Aggregation, Reusable, Object, Interface

计算机的出现,在运算速度和信息存储量上都为人类带来了突破性进展。为了更好地控制它、应用它、发展它,需要有一种抽象介质存在于机器指令和自然语言之间——汇编语言和各种高级语言。应用这些语言,还需一定的开发方法,从面向过程的结构化程序设计到面向对象程序设计方法,继而提出了组件的概念。

"组件"技术的兴起可以说是应人们对可重用、软件标准化的要求,藉面向对象思想及其支持语言(如C++)的日益成熟,再加上各商业公司的推波助澜,真正使软件重用概念复兴起来,并且在组件领域较为成功的是:SUN 的 JavaBeans,OMG 的 CORBA 以及 Microsoft 的 OLE/COM。

微软公司的 OLE/COM 技术,随着微软的 Develop Studio 和 Office 系列的普及,流传渐广。OLE/COM 以标准化、可重用性较好而著称,但其中存在着这样一个问题:当一个组件的对象 A 具体实现了一个接口 I,若再想对此接口 I 进行扩充,而且又要利用对象 A 已实现了的那些接口 I 中的方法,就会产生不合乎规范的一些问题,问题的根源是由于各接口必须直接或间接继承 IUnknown 所引起的。

本文将讨论 COM 技术对于组件可重用的支持以及用于支持多继承的聚合技术等,并针对这些技术引起的扩充性不足问题,提出一些避免、改进的方法。

1. COM(Component Object Model)组件技术

随着软件规模、复杂性的不断增大,"软件重用"日益受到人们的重视。1968 年 McIlroy 在 NATO 软件工程会议上首次提出了软件重用的思想。1983 年 Freeman 又进一步拓广了软件重用的概念,指出可重用的组件不仅可以是源代码片段,还可以是模块设计结构、规格说明和文档等,而且不仅可按组装方式重用,还可按模式重用。目前,组件是软件重用的基本单位。组件具有标准化、模块化强的特性,易于重用,相应地,要求组件的编写制造也要符合一定的规范,使其更具一般性、更高效、更易使用。

COM 就是这样的一种协议规范,它是由 Microsoft、Digital Equipment 和许多其他公司支持的一个"工业标准"软件架构。COM 为 Windows 提供了统一的、可扩充的面向对象的通讯协议。COM 提供了一个实现 OLE 的框架,也为开发一套新软件提出了一种全新的设计思想。

COM 这种协议建立了一个软件模块同另一个软件模块之间的连接,然后再将其描述出来;当这种连接建立起来之后,两个模块之间就可以通过称为"接口"的机制来进行通讯。接口可被理解成两个软件模块之间达成的协议,彼此通过接口的调用知道对方能完成什么功能。只要接口的定义不变,接口的对象所能提供的方法就能被调用。

在 COM 组件的各种要素之中,IUnknown 接口是最重要的,因为任何一个接口都从这里直接或间接继承下来。这个接口只有 3 个纯虚函数有待实现:QueryInterface( )、AddRef( )、Release( )。QueryInterface 会根据参数 ID 值查找正确的、已实现的接口(包括本组件对象内的所有接口)的入口,以便进一步得到该接口中的方法的服务。AddRef/Release 则通过引用计数以求有效管理与释放内存。

COM 是一种在二进制层面上的规范,只要所定义的接口中的函数集合理,接口 ID 符合 128 位 ASCII 字

符子集规范，并且提供一个类来实现各接口的虚函数，再把这个类的类标识 ClassID 和接口标识 InterfaceID 放到大家都公认的地方（如 windows 注册表），就大功告成了。其中最自由的部分莫过于接口各函数的实现，当然各接口的定义和存在与否（除了 IUnknown）也是可以定制的。

在 COM 中，对于非接口类（不是纯虚类）的实现没有继承的概念，因为 COM 的核心是接口，从上面的叙述可知，组件的定制、使用的桥梁是接口。COM 的目标是在同一类中，各接口可以通过 QI（QueryInterface）互相找得到。

2. COM 的可重用性

COM 以接口为中心，尤其是以 IUnknown 为根，决定了它需要其它的机制来代替对"已实现某接口的类"的继承。这是针对那些"已实现接口的类"的多继承（单继承没有意义），对于各接口（纯虚类）的多继承 COM 本来就必须支持的（不过是多了一些纯虚方法而已）。COM 使用包容和聚合技术完成这种多继承。

2.1 包容（containment）

COM 中，可以称将要实现的类为外部类，已经实现的类称为内部类。包容将内部类放在外部类的实现中，外部类向外提供的那些接口中，如果能自己实现的，就象往常一样处理；不想自己实现的接口，需要套用另外封装的基类，则将这些接口委托给基类的相应接口。这时，接口之间通过一下方法互访，以达到一致性的目的。

外部类的接口想访问内部类的接口是没有问题的，如图 1 中，外部对象 CEditPrintObj 肯定知道自己有一个内嵌的对象 CPrintObj，内部对象则只要用一个数据成员记录外部对象的指针，以供返回。下面给出一个以包容实现重用的类的例子。

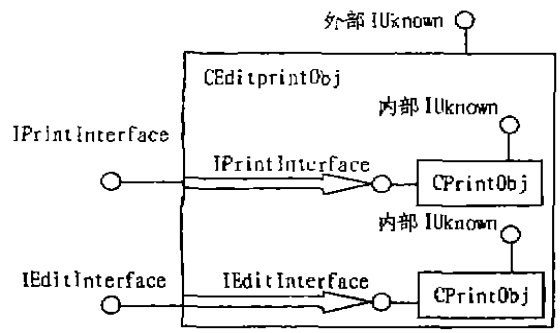

图 1 包容实例 CEditprintObj

```
class CEditPrintObj// 外部对象的类
{
public:
    CEditPrintObj();
    HRESULT QueryInterface(REFIID iid,void** ppv);
    ULONG AddRef();
    ULONG Release();
    DWORD m_dwRef;
    class CPrintObj:public IPrintInterface
    {
    public:
        CEditPrintObj* m_pParent;
            // 指向上层外部对象
        virtual HRESULT QueryInterface(REFIID iid,
                void** ppvObj);
        virtual ULONG AddRef();
        virtual ULONG Release();
    }m_printObj;// 内部对象
    class CEditObj:public IEditInterface
    {
    public:
        CEditPrintObj* m_pParent;
        virtual ULONG QueryInterface(REFIID iid,void**
            ppvObj);
        virtual ULONG AddRef();
        virtual ULONG Release();
    }m_editObj;// 内部对象
};
```

从中可以看出，大多数 IUnknown 实现全集中在 CEditPrintObj，而不是 CEditPrintObj::CEditObj 或 CEditPrintObj::CPrintObj 中，减少了冗余。即所有的 QI 都有同样的行为。从 IUnknown 接口，或者说不论调用哪一个接口的 QI 都能互访。

由外而内：通过成员对象 m_printObj、m_editObj 来访问；由内而外：内部对象自身数据成员 m_pParent 提供指针。这种方法其实在本例中没体现出什么优越性，2 个内部类都有关于 IUnknown 三个函数的重写，又增添数据成员 m_pPerent，但如果还有各自其它的处理函数如 Edit-tools-Modify 之类，而又能被其它的如另一个类 CEditRead 所嵌套，就很有价值了。

2.2 聚合（aggregation）

聚合实现的同样是多继承的内容，但表现方法不一样，包容的内部对象一般不被外部对象直接调用，是供外部类来使用的，而聚合的两个类是独立的，都可以直接被客户端直接调用，是聚合和被聚合而已，当然前提是被聚合类要可被聚合。即内外交通顺畅的措施，"可被聚合"就是提供出一套能够顺利完成 QI 功能的机制。

从图 2 可知，被聚合的类仍然是嵌在聚合类内部。外部访问内部，是通过内嵌的被聚合类的 IUnknown 指针来达到的；内部访问外部，也是用了一个公有型的数据成员，专门记载被哪一个类所聚合了。下例中定义了 CSpaceShip 类，它对外公布的是接口 IUnknown、IVisual、IMotion 及其方法，但它只实现了 IVisual 中的各个方法，对于 IMotion 根本未予考虑，它借用（聚合）了一个已存在类 COrbiter，直接使用 COrbiter 的 IMotion 接口的实现为自己服务。下面是在 MFC（Microsoft Foundation Class）机制下的部分实现：

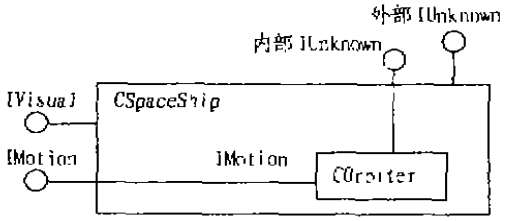

图 2 聚合实例 CSpaceShip

```
class CSpaceShip:
      public CCmdTarget
{
public:
    CSpaceShip();
protected:
   LPUNKNOWN m_lpAggrInner;
   virtual BOOL OnCreateAggregates();
DECLARE_INTERFACE_MAP()
      //进行接口映射的宏
      //"native" interface part macros may be used here
};
CSpaceShip:: CSpaceShip()
{
    m_lpAggrInner=NULL;
}
BOOL CSpaceShip::OnCreateAggregates()
{
    //wire up aggregate with correct controlling unknown
    m_lpAggrInner = CoCreateInstance ( CID_ORBITER,
    GetControllingUnknown(),CLSCTX_INPROC_SERV-
    ER,IID_IMotion,(LPVOID * )&m_lpAggrInner);
      if(m_lpAggrInner==NULL)
      return FALSE;
    //optionally,create other aggregate objects here
    return TRUE;
}
```

这里没有 IUnknown，是 CCmdTarget 的成员函数 ExternalQueryInterface、ExternalAddRef、ExternalRelease 得到实现下来的。

系统完成下面这种映射，会自动在找不到所需接口时，从自己聚合的对象中去搜寻；反之，COrbiter 必须在自己构造函数中将聚合机制"使能"(enable)，如使用 CCmdTarget::EnableAggregation()函数。下面列出了聚合主要利用的 INTERFACE_AGGREGATE()宏(其它宏虽未予详细说明，也很重要)：

```
BEGIN_INTERFACE_MAP(CSpaceShip,CCmdTarget)
   //native"INTERFACE_PART"entries go here
INTERFACE_AGGREGATE(CSpaceShip,m_lpAggrInner)
END_INTERFACE_MAP()
```

本例中，COrbiter 类嵌在 CSpaceShip 的内部。先看从外部进入内部的情况：假定客户获得了 CSpaceShip 的 IVisual 指针，将调用 QI 接口获取 IMotion 指针，这时若使用外部的 IUnknown 去查，只能得到空指针，因为 CSpaerShip 类并不支持 IMotion。继而，CSpaceShip 类通过 m_lpAggrInner 记录了内部的 IUnknown(可以通过内嵌的 COrbiter 对象指针获得)；继而，CSpaceShip 可以用该指针获得 COrbiter 对象的 IMotion 指针。

再看从内部到达外部的情况：假定客户获得了 IMotion 指针，将调用 QI 接口以获取 IVisual。这时，COrbiter 类的一个数据成员将指向"控制 unknown"(controlling unknown)，即指向 CSpaceShip 的外部 IUnknown 指针。在 MFC 的实现上，一个实现接口的类通常继承自 CCmdTarget 类，此类中有一个公有型数据成员 m_pOuterUnknown，当本对象被外部对象聚合的话，用来保存外部对象的 IUnknown 指针。相应地，可以找到外部的 IVisual 接口。

现在我们已经了解了 COM 用于实现多继承的两种手段，似乎很完善，尤其是"聚合"几乎有收放自如的意味，外部对象直接把内部对象的一些接口传向了真正的客户；聚合又是递归嵌套的，如上例中 GetControllingUnknown()函数可以将本类的聚合宿主的 IUnknown 指针传递到下一层被聚合对象中去，使最内层的被聚合类也能与"聚合宿主"联系。除了记录下一层的 IUnknown 指针外，不必多定义些什么，充分体现了以接口为中心、IUnknown 为接口根的干净利落，但恰是如此，在可重用性和可扩充性上带来了一些问题。

3. 以接口为中心的 COM 聚合带来的问题

从上述可以看出，在一般情况下，聚合机制做得很好，一组对象可共同工作在具有良定义的环境中(每个对象的类是独立时)，并可作为单个对象出现在其它软件组件中(被其它类聚合了)。当保持所有对象间清晰关系时，聚合提供了代码重用，并避免了实现多继承中可能会出现的一些危险(如冗余、重名)，但聚合相对有些复杂，因为它毕竟要实现与那些运行时才装载的对象通讯，即遵守"使用者必须能从任一个接口转到任一个另外的接口"的约束条件，IUnknown 的基本方法之一 QI 应当返回当前对象所支持的任何接口，这样就抹杀了"简单实现继承"的可能性，为什么这样说呢？见下例：

①现有一个类 ScanObj，实现了接口 IScan，用于完成扫描仪的功能。这很容易做到，如用 C++可以先定义一个抽象类 IScan(其中所有的方法都是纯虚的)，再定义一个具体的类 ScanObj(这个类继承了 IScan，并实现了它所有的方法)。

②现在有个需要，要扩展 ScanObj 这个类，使其支持另一个接口 IPrinter，因为开发了一种新的产品，将扫描仪和打印机合二为一。

在 C++中，这个"扩充需求"及同时要满足对原有"已实现类"ScanObj 的重用可谓小事一桩：只要定义一个类 ScanPrinter，使它继承自 ScanObj 和 IPrinter，

这个新类不仅支持原有的IScan(即支持其所有方法),而且连带给出了IScan的实现—因为ScanPrinter继承自ScanObj;同时这个类还支持接口IPrinter并也将给出IPrinter接口中各方法的实现。

对于C++很容易的事,到了COM中,就不那么方便了。如果你不希望去改动原来的类的实现的话。上面已经反复申明,COM要想实现上述对象与接口的综合继承,必须要实现能够用IScan接口的QI方法获得IPrinter接口(更细致地说是通过VTBL获得相应接口指针)。需要注意的是:ScanObj提供了IScan所有方法的实现,包括QI方法,其中当然不会提供任何有关IPrinter的方法入口,因为在扫描仪类ScanObj建造之初不会想到将来还要加上打印机IPrinter功能,它当然不会费劲地为访问IPrinter保留些什么。

不过,还有一种可能,就会从IPrinter入手,实现该接口,把ScanObj聚合进来(当ScanObj是可被聚合的时,即它具有公有的指向上层的指针成员,并且在本身的构造函数中已被enable)。而如果IPrinter也是一个许久以前被实现了的类,则需要它也是可被聚合的,最后在另外的新接口中进行2次聚合,将ScanObj和PrinterObj这2个类合并进来。

问题是,如果这些对象编造年代久远,没有提供被聚合入口,或者集成多个接口时没有MFC这样的现成机制,就很麻烦了,需要对这些类的实现动些干戈,重用的精髓无从体现。

怎样解决呢?如果以对象为中心使用"组件"情况就不一样了。注意,这儿的对象保留了接口的所有长处、特点,该对象所对应的类是在COM规范下的新的意义上的类。可以说,接口是重要的,但新的意义上的对象更重要,因为作为接口的实现者,不论用何种语言,都不得不涉及到对象。如上述例子中,描述了2个接口:IScan和IPrinter,还描述了2个类:ScanObj和ScanPrinter(相应有两者实例化的对象)。然而,这种很抽象的概念完全不被COM的用户所意识到,使用者只能看见接口,如下例:

```
hErr = CoCreateInstance (CLSID_IBeep, NULL, CLSCTX_
       INPROC_SERVER, IID_IUnknown, (LPVOID
       FAR*)&pUnk);
pUnk-> QueryInterface (IID_IBEEP, (LPVOID FAR*)
       &pBeep);
pBeep->DoBeep(-1); //DoBeep是IBeep接口的一个方法
```

但对象是确实存在的,不论直接地存在于用户的机器上,还是通过代理进程来远程调用。而且,即便在COM中,也只有当2个接口共享同一个底层对象时,才能从一个接口跳到另一个接口。比如通过一个接口改变一个对象的状态,而通过另一个接口检查一下这个状态。很明显,这2个行为针对的是同一个对象。所以,如果COM有了关于对象的明确说法可能会更简单。

如用C++代码来考虑调用组件时,可用以对象为中心的思想:

```
CoObject* obj=CoCreateInstance(IID_ScanPrinterObj);
    //通过InterfaceID找到相应组件的类并实例化
IScan*s can=obj→QueryInterface(IID_SCAN); //通过对象
    obj获得接口IScan的指针
scan→ScanMethod(); //使用IScan接口的方法
IPrinter* printer=obj→QueryInterface(IID_PRINTER);
    //通过对象obj获得接口IPrinter的指针
printer→PrinterMethod(); //使用IPrinter接口的方法
delete obj;
```

这里特意省略了所有的错误检查和引用计数,正式应用中加入就行了。可见,这里没有从一个接口到另一个接口的切换,也没有用到IUnknown。扩充工作为:从ScanObj和IPrinter那里进行继承;实现IPrinter的各个方法;当然,这要重写CoObject的QI方法。这里假设所有对象继承自抽象类CoObject,然后重写其QI方法。调用QI时指出相应的IID接口指针即可。需要指出的是,这与每个接口从IUnknown继承时不一样,从IUnknow继承时是不会考虑已实现了的对象的方法的。

在http://www.relisoft.com站点上提供的Smart Ole考虑了包括引用计数等方面的更为完善的方法,它用IDispatch代替了IUnknown,又利用模板类型简化书写,在此恕不详述。

结束语 本文讨论了COM中对于已实现对象的可重用和可扩充性存在的一些问题,为了阐明观点,较为详细地讲述了COM及其聚合等的概念。COM既然是一种组件规范,要成为一种标准,编写者、使用者都应该注意其可重用及扩充问题,如果继续以接口为中心,应当注意对已实现对象的扩充及考虑对今后编写组件提供可重用基础;否则还不如在使用者一方约定好以对象为中心。但这种以对象为中心,决不是放弃了"接口"机制的倒退,而是更高层次上的使用接口。与C++相比,COM有ID机制又是业界规范,客户可直接访问、利用已存在的组件,所以可以进一步支持分布式结构,相应地导致了DCOM的发展。

参考文献

1 Kruglinski D J. Inside Visual C++,4th Edition,1997
2 Waters B. Mastering OLE2,1995
3 Chavez A, et al. Software Component Licensing. IEEE Software,1998,15(5)
4 Kotula J. Using Pattern To Create Component Documentation. IEEE Software,1998,15(2)
5 秦大勇,等.面向对象可视化重用构件的语言动态特性研究.计算机科学,1998,25(4)
6 Available at:http://www.relisoft.com/olerant.htm
7 Available at:http://www.relisoft.com/ole.htm
8 Available at:http://www.relisoft.com/auto.htm
9 微软 Technical Note # 38,(MFC/OLE IUnknown Implementation)