目 录

[一. 对象与接口 2](#_Toc15484928)

[二. COM应用模型及COM库 7](#_Toc15484929)

[三. 类厂对象 11](#_Toc15484930)

[四. VC++调用COM组件的方法 15](#_Toc15484931)

[五. 使用MFC开发COM组件 20](#_Toc15484932)

[六. 使用ATL开发COM组件 23](#_Toc15484933)

[七. 扩展:com组件和一般dll的区别 28](#_Toc15484934)

# 术语介绍

* IDL

IDL是接口定义语言。

* MIDL

MIDL是Microsoft的IDL编译器。在用IDL对接口和组件进行了描述后，可以用MIDL进行编译，生成相应的代理和存根DLL的C代码

# 对象与接口

一.前言

       在COM规范中，最基本的两个要素就是对象与接口，因为COM就是由这两者来共同实现的。COM对象在组件中是被封装起来的，客户代码只能通过接口来访问COM对象并享受其服务，由于客户与COM直接打交道的是COM接口，所以COM接口是COM最关键的要素。

二.COM对象

         COM对象其实就类似于C++中的对象，也就是说某个类的实例，包含了一组数据和操作。在COM模型中，COM对象的位置对于客户来说是透明的，即客户代码不需要直接初始化一个COM对象，而是COM库通过一个全局标识码GUID去对其进行初始化工作。GUID是一个128位的标识符，基本保证了COM对象的唯一性，另外COM接口也是用GUID来标识的。

封装特性

        COM对象的封装特性与C++的封装特性有所不同，COM对象的成员数据是以组件模块为边界，对于客户代码来说是看不见的，只能通过接口的函数去访问属性；而C++对象的封装特性则是语义上的封装，对于客户代码来说是可见的（一个C++类有什么样的成员变量是可见的，但是一般都是private或者pretected，无法直接访问）。

可重用性

          COM对象的可重用性在表现在聚合与包容，使得一个对象可以完全使用另一个对象的所有功能；而C++的重用性表现在继承上，则子类对象完全继承父类对象的属性与方法。聚合与包容可以使COM对象A能够完全重用COM对象B的功能，就像A实现了B的功能一样，并且在B的版本更新之后，A会动态地调用新版本的B而不需要重新编译、设置A，因此COM对象的重用是动态的。而C++对象则更多通过继承来实现源代码一级的重用，若更新了父类的定义，则子类必须重新编译，编译出来的可执行文件是在同一模块内的，重用性只体现在模块内部。而实际上在COM开发中同时使用了两者，即在COM内部使用了C++的代码级重用，在COM外部使用了组件重用。

多态性

          C++的多态性是其面向对象特性中最重要的一环，它是通过虚函数的机制来实现的。而COM对象也具有多态性，它是通过COM接口来实现的，即一个COM允许一个对象同时实现多个接口，因此，一个COM对象的多态性是由其接口实现的。

三.COM接口

          COM接口通常是一组以函数的逻辑集合，其命名一般以“I"为前缀，并且继承IUnKnown接口。COM对象可以提供多个COM接口，每个接口提供不同的服务，因此COM接口与COM对象一样，都是用GUID来标识的，客户通过GUID来获取接口指针，再通过接口指针获取对应的服务。

二进制特性

        接口规范并不建立在任何编程语言的基础上，而是在二进制一级的标准，任何语言只要有足够的表达能力对接口进行描述即可开发COM的应用（C++，C#，VB）。

接口不变性

       接口是客户与COM对象之间的桥梁，其重要性不言而喻，若接口总是变化，会使客户代码也跟着变化，这对于应用的开发是很不利的，所以接口一般是不变的。

继承特性

        COM接口的不变性，不代表接口不能拓展，可以通过接口继承的方式来进行拓展，与C++类的多继承相比，接口继承只允许单继承，并且继承的是接口函数的声明部分，并不包括接口的实现。

多态性

       多态是OOP的重要特点，对于COM来说也一样，而COM对象的多态性是通过COM接口的多态来体现的。

        IUnknown接口是COM的核心，因为所有其他的COM接口都必须从IUnknown继承。它包含三个接口函数：QueryInterface、AddRef和Release，其中QueryInterface用于接口查询，从COM对象的一个接口获得另一个接口，一个对象可能实现了多个接口，这样就可以通过QueryInterface在对象多个接口之间跳转从而获得多个接口提供的服务；AddRef与Release则用于管理COM对象的生命周期，当COM对象不再使用时需要释放，因此COM使用了引用计数的方法来对对象进行管理，当有一个用户获得接口指针后调用AddRef将引用计数加1，相反，当一个用户用完接口指针后就调用Release来使引用计数减1，这样当引用计数为0时，COM对象就可以从内存中释放。由于IUnknown提供了接口查询与生命周期控制两个功能，因此COM的每个接口都应该继承于它。

接口IUnknown的定义：

class IUnknown{

public:

virtual HRESULT \_\_stdcall QueryInterface(const IID& iid, void\*\* ppv) = 0;

virtual ULONG \_\_stdcall AddRef() = 0;

virtual ULONG \_\_stdcall Release() = 0;

};

可以看出IUnknown实质上就是一个含有纯虚函数的抽象类。

# COM应用模型及COM库

C/S模型

        COM对象与客户程序之间是基于C/S（客户端/服务器）的模型在运行的，COM服务器是组件对象的容器，由组件对象向COM客户提供服务，所以COM的应用模型继承了C/S模型的优点：高可靠与稳定性、很强的拓展性并且性能高等。

包容与聚合

        COM不是简单的C/S模型，有时客户可以反过来提供服务，COM对象提供服务的同时也可以向其他组件寻求服务，所以COM对象可能是一个服务方的同时也是客户，而C/S在这里只是相对的。COM提供了两种对象的重用机制，分别是包容和聚合，对于客户而言，他只知道对象B的存在而不知道对象A，而对象B可能在实现一些服务时调用了对象A的服务。两者的区别在于，包容是指客户调用B的服务时，由B直接调用A的服务，并把结果返回给客户，而聚合是指客户调用某个服务的时候，B直接把A的指针返回给客户去操作，但客户不知道A的存在，但他是直接调用A的。

组件类型

        根据COM客户与COM对象是否运行在同一进程空间内，COM服务器可以分为进程内组件与进程外组件，而进程外组件又分为本地组件及远程组件。

        进程内组件：组件直接加载到客户的进程地址中，通常是DLL的形式存在。

        本地组件：组件程序与客户程序在同一台机器上，但是组件程序是以独立的形势存在，通常是EXE。

        远程组件：组件程序与客户在不同机器上，可以使DLL也可以使EXE。

          尽管COM分为几种进程模型，但是这种区别对于客户来说是透明，客户调用哪类组件都是一样的操作，因为COM库会处理这些差异，它会负责定位组件，管理组件对象的创建与销毁，还有对象与客户之间的通信。尽管对用户来说，三种进程模型是透明的，但是选用的时候也要慎重。进程内模型的优点是效率高，但由于进程内组件是加载到客户进程地址的，所以组件的不稳定可能会使客户程序崩溃；而进程外组件的优点是稳定性强，组件进程不牵涉到客户进程，并且一个组件进程可以服务于多个客户，但是开销大、效率相对低点。

COM库

        COM库是COM组件与客户之间的桥梁，在组件的创建、对象和内存管理方面起着至关重要的作用。每个程序在使用COM之前必须初始化，否则对COM库函数的调用都无效。

COM的初始化函数原型：

HRESULT CoInitialize(Imalloc\* pMalloc);

参数是一个内存管理器的指针，通常设置为NULL，使用默认的管理器。

返回值可能是：S\_OK、S\_FALSE、S\_UNEXPECTED。

同理，在使用完COM之后要释放资源，调用：

void CoUninitialize(void);

凡是调用CoInitialize并返回S\_OK的必须调用CoUninitialize以保证资源的有效利用。

COM库常用函数：

初始化函数

函数 描述

CoBuildVersion 获取COM版本号

CoInitialize 初始化COM库

CoUninitialize 终止COM库功能服务

CoFreeUnuseLibraries 释放进程所有不再使用的组件程序

GUID相关函数

函数 描述

IsEqualGUID 判断两个GUID是否一样

IsEqualIID 判断两个IID是否一样

IsEqualCLSID 判断两个CLSID是否一样

CLSIDFromProgID 把字符串形式的对象识别符转化为CLSID

StringFromCLSID 将CLSID转换为字符串

IIDFromString 将字符串形式的接口标识转换为IID

StringFromIID 将IID转换为字符串形式

StringFromGUID2 将GUID转换为字符串形式

对象创建

函数 描述

CoGetClassObject 获取对象的类厂

CoCreateInstance 创建COM对象

CoCreateInstanceEx 创建COM对象，并可指定多个接口或远程对象

CoRegisterClassObject 注册一个对象以便其他对象可以找到

CoRevokeClassObject 对象反注册

CoDisconnectObject 断开其他应用于对象的连接

内存管理函数

函数 描述

CoTaskMemAlloc 内存分配

CoTaskMemRealloc 内存重新分配

CoTaskMemFree 内存释放

CoGetMalloc 获取COM库的内存管理器接口

COM与注册表

        COM客户是通过系统的注册表去找到COM组件的位置，然后再去加载它的，可以在WIN+R，然后输入regedit打开注册表，找到注册表HKEY\_CLASSED\_ROOT节点下的CLSID子键，它列出了当前机器上的所有组件信息。如果是进程内组件，CLSID下会有InprocServer32子键，该子键默认是组件的全路径文件名，而进程外组件的全路径文件名则是在LocalServer32子键中。客户程序请求COM组件服务的时候，就是通过COM库去注册表搜素对应CLSID的组件的全路径文件名，然后完成对象的创建，并把对象的接口指针返回给客户使用。

# 类厂对象

COM类厂对象的实现

        COM客户程序要使用COM对象是通过COM库创建而来的，而实际上COM库是调用COM对象的类厂来创建的。COM类厂对象也是一个COM对象，所以它也从IUnknow继承而来，而它又支持IClassFactory接口：

class IClassFactory:public IUnknow

{

public:

virtual HRESULT CreateInstance(IUnkonwn\* pUnkOuter, REFIID iid, void\*\* ppObject)=0;

virtual HRESULT LockServer(BOOL fLock)=0;

};

        所以，一个普通的类厂应该是这样的：

class CSampleFactory: public IClassFactory

{

public:

CSampleFactory();

HRESULT \_\_stdcall QueryInterface(REFIID riid, void\*\* ppObject);

ULONG \_\_stdcall AddRef();

ULONG \_\_stdcall Release();

HRESULT \_\_stdcall CreateInstance(IUnknown\* pUnkOuter, REFIID riid, void\*\* ppObject);

HRESULT \_\_stdcall LockServer(BOOL fLock);

private:

ULONG m\_dwRefCount;

};

CreateInstance是构造COM对象的函数，通过传入接口的IID，从ppObject输出COM接口指针，而pUnkOuter一般设为NULL，该参数在聚合时起作用。

LockServer是用来控制COM类厂的生命周期的函数，将fLock设为TRUE后，即使组件程序中所有COM对象已释放了，该类厂指针也会一直保存并且有效，当不再需要的时候设为FALSE即可。

         若要使用类厂对象去创建COM对象，首先得创建类厂对象，可以使用库函数CoGetClassObject来创建COM类的类厂，若找到的COM对象是进程内组件，则使用DLL导出函数DllGetClassObject函数创建类厂，然后将对象指针传出。

STDAPI DLLGetClassObject(REFCLSID rclsid, REFIID riid, void\*\* ppObject)

{

if(rclsid == CLSID\_SAMPLE)

{

CSampleFactory\* csf = new CSampleFactory();

if(FAILED(csf->QueryInterface(riid, ppObject)))

{

delete pFactory;

\*ppObject = NULL;

return E\_INVALIDARG;

}

}

return NO\_ERROR;

}

        若创建的是进程外组件，则要分别调用CoRegisterClassObject和CoRevokeClassObject去注册和反注册类厂对象才能正常地创建、销毁COM对象。

COM自动注册

        在cmd窗体下，使用命令

                regsvr32 d:\sample.dll

        对sample.dll进行注册，这时会调用该dll导出的DllRegisterServer。

        另外，使用

                regsvr32 /u d:\sample.dll

        对其进行反注册，同样会调用导出函数DllUnregisterServer。

        以上的命令是对进程内组件有效，而对进程外组件不必使用以上命令，由于自身是可执行程序，所以在运行时一般会自动调用DllRegisterServer和DllUnregisterServer。

COM自动卸载

         客户端程序调用CoFreeUnusedLibraries来释放组件，但是释放组件需要满足：组件中的对象数目为0和类厂的锁计数为0，要知道是否满足以上条件，需要导出函数DllCanUnloadNow，该函数通过判断组件对象的引用计数来判断是否可以释放组件。

# VC++调用COM组件的方法

ActiveX 控件

         对于ActiveX控件，可以在Visual Studio中使用ClassWizard的方法引入，在MFC项目中选择菜单“项目”——> “添加类”——>“ActiveX控件中的MFC类”，然后可以分别从注册表和文件两种方式中选择自己想要的组件。选择注册表中的方式获取的都是在本系统中注册了的组件，而文件则是从磁盘目录下获取COM组件。



         如果想为现有程序添加对ActiveX控件的支持，可以在应用程序的InitInstance函数中添加：

AfxEnableControlContainer

         并在staafx.h中添加：

#include <afxdisp.h>

代码组件

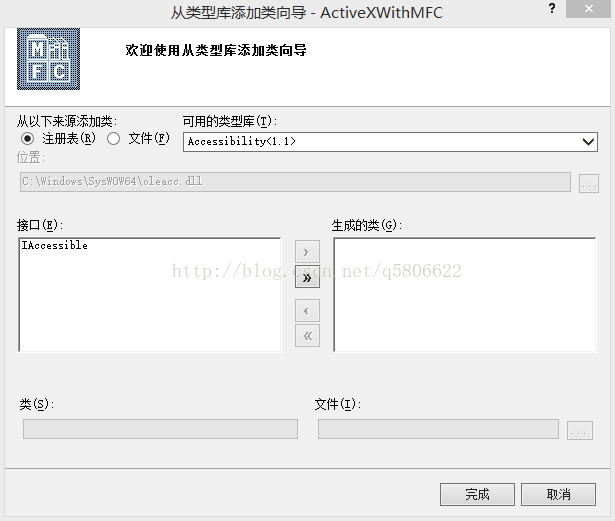
            这类组件就是普通的没有界面的COM组件，需要再客户端程序的代码中进行引入。一般有如下几种方法：

           1. 使用MFC OLE 的ClassWizard：

           在MFC项目的应用程序类（\*APP）中的InitInstance中添加

AfxOleInit();

           然后，选择菜单“项目”——> “添加类”——>“TypeLib中的MFC类”



            选择自己想要的类型库，点击“完成”后会创建自己对应类型库的组件包装类，然后在代码中调用组件包装类类对象的CreateDispatch和ReleaseDispatch来创建和销毁对象。

2.使用COM库函数

在应用程序类的InitInstance函数的开始和结束处分别添加：

BOOL CSampleApp::InitInstance()

{

CoInitialize(NULL);

...

CoUninitialize();

return FALSE;

}

          然后，在对话框类中包含组件的头文件：

#include "COMSample.h"

#include "COMSample\_i.c"

         在对话框类中定义：

protected:

ISample\* pSample;

         之后，在对话框类的OnInitDialog中创建COM对象：

pSample = NULL;

HRESULT hr = CoCreateInstance(CLSID\_Sample, NULL , CLSCTX\_INPROC\_SERVER, IID\_ISample, reinterpret\_cast<void\*>(&pSample));

         其中，CoCreateInstance第一个和第四个参数分别是类ID和接口ID，需要从COMSample\_i.c中查找，所以当接口比较多的时候是比较麻烦的。

         最后，在OnClose中释放COM对象：

if(pSample != NULL)

pSample->Release();

        3.使用import指令

        使用import可以对类型库文件、exe、dll进行类型导入，从而读出组件的类型信息并创建接口的包装类。

         import指令：

#import "COMSample.tlb"

        或者：

#import "COMSample.dll"

         另外，可以把组件的命名空间带入，在import指令的最后添加：no\_namespace

         然后会自动生成跟类型库同名的两个后缀分别是“TLI”和“TLH”的文件，其中TLH文件中包含了对组件每个接口的智能指针的定义，所以可以使用下面的方式使用组件：

ISamplePtr iSamplePtr(\_\_uuidof(Sample));

         或者：

ISamplePtr iSamplePtr = NULL;

HRESULT hr = S\_OK;

hr = iSamplePtr.CreateInstance(\_\_uuidof(Sample));

         其中，第二种方法可以对hr进行检测从而判断错误。

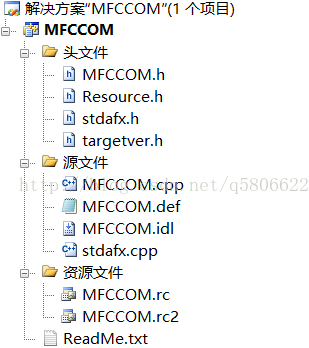
# 使用MFC开发COM组件

使用MFC创建一个进程内组件

          这里使用的VS2008，新建一个MFC DLL项目，项目名称为“MFCCOM”，点击“确定”后进入MFC DLL向导，如下图所示：



            DLL类型选择“使用共享的MFC DLL的规则DLL”，附加功能勾选“自动化”（自动化是一种支持弱类型语言调用的特性），然后点击“完成”即可，生成的项目文件目录如下图：



         生成的项目里包含了一个DLL的应用类和几个导出函数的定义，还有def文件和idl文件的生成。

         为了开发COM，需要定义自己所需的COM对象类，这可以使用类向导来完成。通过点击菜单的“项目”——》“添加类”，选择“MFC类”，点击添加：



          这里基类选择CCmdTarget，并勾选了自动化，这样这个类就是一个支持IDispatch接口的COM类了。

           之后，只需添加项目自身的接口函数到类中即可，而且每个接口函数必须是以STDMETHODIMP作为返回值。

           例如：

STDMETHODIMP Test(BSTR\* result);

# 使用ATL开发COM组件

使用ATL开发COM组件

          使用ATL开发COM组件之前，要对几个重要的类进行说明：

          CComModule

          这是ATL的中心模块类，它的作用有点像MFC中的CWinApp，但没有后者那么庞大。它和它导出的类主要负责处理引用技术、在本地服务器内的类厂注册以及管理自动注册和自动反注册的注册表管理程序间的协调。若要创建进程内组件，只需直接使用CComModule，但如果是创建本地组件，则需要派生CComModule。

          CComObjectRoot 和 CComObjectRootEx

          他们的基本功能是处理组件的引用计数并对聚合提供支持，其中后者是一个类模板，模板参数是线程模型，而前者是后者取一个默认的线程模型的实例。

          CComCoClass

          该类主要负责类厂的定义，并提供两个标准方法取得对象的CLSID和说明。

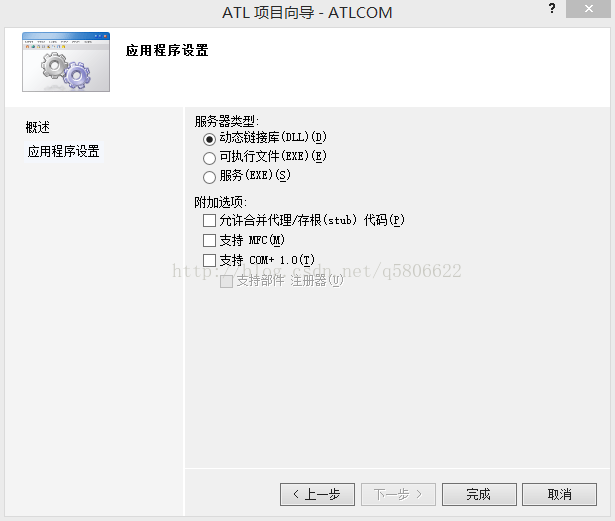
          CComObject

          它用于实现IUnknown接口，这个类比较特殊，因为它是一个模板类，需要从我们自定义的类继承，模板参数是自定义类的类名，它的代码不需要我们编写，由其内宏自动生成。

          ATL创建的组件类必须从CComObjectRoot和CComCoClass派生出来，此外还必须从自定义组件类派生CComObject类。

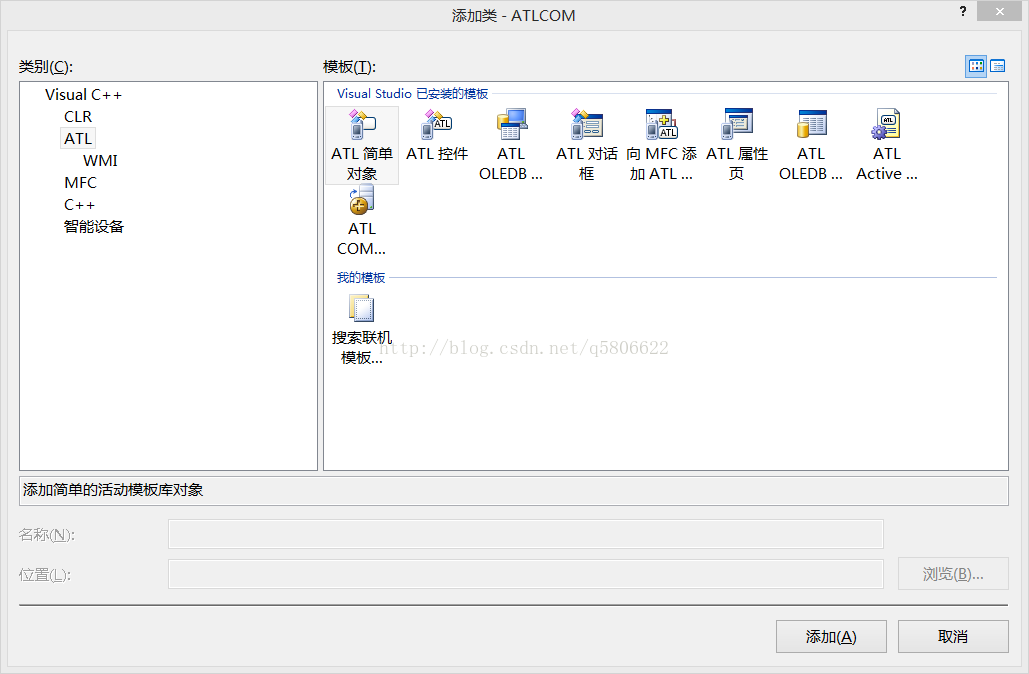
在VS2008中开发ATL

          创建一个ATL项目，名称叫ATLCOM，点击“确定”后，弹出项目配置向导窗体：



           保持默认设置，点击“完成”，创建一个进程内组件。

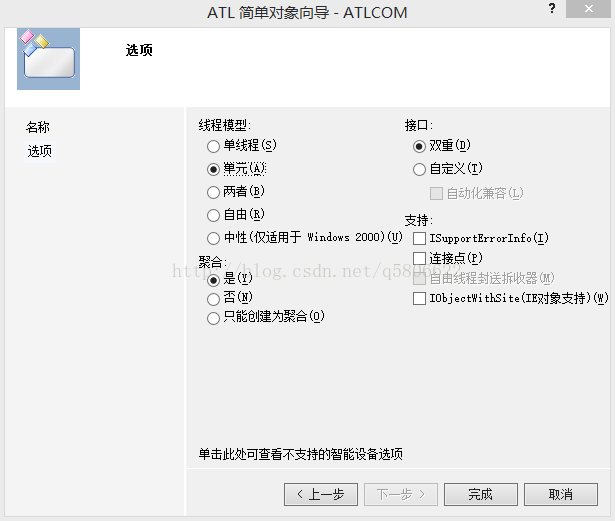
           然后，选择菜单“项目”——》“添加类”，选择ATL简单对象



         点击“添加”，然后按照自己的需要，定义组件类的简称，向导会自动补全其他信息。



          点击下一步，进入ATL选项设置，这里保持默认设置点击“完成”即可成功创建COM组件类。



         然后打开类视图，可以看到ATLCOM下新增了CSample类和ISample接口：



          现在，可以对ISample点击右键——》添加方法（添加属性）来丰富接口了，然后在CSample内实现相应的的方法（属性）即可。

# 扩展:com组件和一般dll的区别

这阵子在想一个需要利用com组件的小程序怎么做，突然想起上次去面试的时候考官问过autocad开发时为什么要利用com，而不采用一般的dll呢？

到google上查了一下，许多人也问了一样的问题：） 用com来写程序要比普通的dll麻烦一些，但是带来的好处也大很多，尤其是在开发像autocad这样大型软件的时候，需要跨区域来协同工作。 “学习COM，首先要知道COM的目的是什么，它解决了一个什么样的问题，在此基础上，MS提供了各式各样的可扩展的COM服务，COM编程接口。这些服务和接口恐怕一个程序员一辈子都不可能通晓，因为发展是如此之快。大家也没有必要通晓这些接口，就象对待大量的WIN API的态度，用到时再查帮助。

COM的各种努力都是在规定了一种二进制交互的协议。说起来简单，做起来相当复杂，要使使用不同语言编写的客户能够使用任意语言编写的服务程序谈何容易！这里说语言还是把问题简单化了，因为每种语言还有各式各样的编译器，不同编译器出来的二进制代码如何交互？

就拿DLL来说，DLL是对静态连接的一种改进，带来了更细的开发分工，也带来了很多问题，

其中就有二进制如何交互的问题。这个问题当DLL输出类时更加突出。COM为解决此问题提出了极负创意的解决方案，不仅如此，更进一步引申，提出了如何跨 网络的交互。然后，针对internet服务器的开发提出COM+。COM体系中融合了多种经典的设计模式，可以说是一种更加精干的C++。

COM博大精深，若干大师仍须半年的mental fog, 方能有豁然开朗之时。我们这些晚辈，还需多加修炼才对，但决不止是COM“

1、COM组件以接口对功能分类，便于组织；DLL特别是大的DLL，函数一大堆，难以组织；

2、COM组件便于升级维护，功能扩充，只需添加接口就行；DLL升级困难，函数不能随意改变；

3、COM创建调用有很好的安全性，DLL没有；

4、COM组件可轻松实现进程间调用，DLL很困难；

5、COM组件可轻松实现分布式调用，DLL不可能；

6、COM组件具有封装、继承、多态的面向对象特征，DLL只有封装； 7、在COM组件的基础上实现了大量功能：ActiveX，OLE等；

Activex、OLE、COM、OCX、DLL之间的区别

熟悉面向对象编程和网络编程的人一定对ActiveX、OLE和 COM/DCOM这些概念不会陌生，但是它们之间究竟是什么样的关系，对许多们还是比较模糊的。在具体介绍它们的关系之间，我们还是先明确组件 （Component）和对象（Object）之间的区别。组件是一个可重用的模块，它是由一组处理过程、数据封装和用户接口组成的业务对象（Rules Object）。组件看起来像对象，但不符合对象的学术定义。它们的主要区别是：

1）组件可以在另一个称为容器（有时也称为承载者或宿主）的应用程序中使用，也可以作为独立过程使用；

2）组件可以由一个类构成，也可以由多个类组成，或者是一个完整的应用程序；

3）组件为模块重用，而对象为代码重用。现在，比较流行的组件模型有COM（Component Objiect Module，对象组件模型）/DCOM（ Distributed COM，分布式对象组件模型）和CORBA（Common Object Request Broker Architecture，公共对象请求代理体系结构）。

到这里，已经出现了与本文相关的主题COM，而CORBA与本文无关，就不作介绍。之所以从组件 与对象的区别说起，是想让大家明确COM和 CORBA是处在整个体系结构的最底层，如果暂时对此还不能理解，不妨继续往下看，最后在回过头看一看就自然明白了。现在开始阐述ActiveX、OLE 和COM的关系。首先，让大家有一个总体的概念，从时间的角度讲，OLE是最早出现的，然后是COM和ActiveX；从体系结构角度讲，OLE和 ActiveX是建立在 COM之上的，所以COM是基础；单从名称角度讲，OLE、ActiveX是两个商标名称，而COM则是一个纯技术名词，这也是大家更多的听说 ActiveX和OLE的原因。既然OLE是最早出现的，那么就从OLE说起，自从Windows操作系统流行以来，“剪贴板”（ Clipboard）首先解决了不同程序间的通信问题（由剪贴板作为数据交换中心，进行复制、粘贴的操作），但是剪贴板传递的都是“死”数据，应用程序开 发者得自行编写、解析数据格式的代码，于是动态数据交换（Dynamic Data Exchange，DDE）的通信协定应运而生，它可以让应用程序之间自动获取彼此的最新数据，但是，解决彼此之间的“数据格式”转换仍然是程序员沉重的 负担。对象的链接与嵌入（Object Linking and Embedded，OLE）的诞生把原来应用程序的数据交换提高到“对象交换”，这样程序间不但获得数据也同样获得彼此的应用程序对象，并且可以直接使用 彼此的数据内容，其实OLE是Microsoft的复合文档技术，它的最初版本只是瞄准复合文档，但在后续版本OLE2中，导入了COM。由此可见， COM是应OLE的需求而诞生的，所以虽然COM是OLE的基础，但OLE的产生却在COM之前。 COM的基本出发点是，让某个软件通过一个通用的机构为另一个软件提供服务。COM是应OLE 的需求而诞生，但它的第一个使用者却是OLE2，所以COM与复合文档间并没有多大的关系，实际上，后来COM就作为与复合文档完全无关的技术，开始被广 泛应用。这样一来， Microsoft就开始“染指”通用平台技术。但是COM并不是产品，它需要一个商标名称。而那时Microsoft的市场专家们已经选用了OLE作为 商标名称，所以使用COM技术的都开始贴上了 OLE的标签。虽然这些技术中的绝大多数与复合文档没有关系。Microsoft的这一做法让人产生这样一个误解OLE是仅指复合文档呢？还是不单单指复 合文档？其实OLE是COM的商标名称，自然不仅仅指复合文档。但Microsoft自己恐怕无法解释清楚，这要花费相当的精力和时间。 于是，随着Internet的发展，在1996年春，Microsoft改变了主意，选择ActiveX作为新的商标名称。ActiveX是指宽松定义 的、基于COM的技术集合，而OLE仍然仅指复合文档。当然， ActiveX最核心的技术还是COM。ActiveX和OLE的最大不同在于，OLE针对的是桌面上应用软件和文件之间的集成，而ActiveX则以提 供进一步的网络应用与用户交互为主。到这里，大家应该对ActiveX、OLE和COM三者的关系有了一个比较明确的认识，COM才是最根本的核心技术， 所以下面的重点COM。让对象模型完全独立于编程语言，这是一个非常新奇的思想。这一点从C++和Java的对象概念上，我们就能有所了解。但所谓COM 对象究竟是什么呢？为了便于理解，可以把COM看作是某种（软件）打包技术，即把它看作是软件的不同部分，按照一定的面向对象的形式，组合成可以交互的过 程和以组支持库。COM对象可以用C++、Java和VB等任意一种语言编写，并可以用DLL或作为不同过程工作的执行文件的形式来实现。使用COM对象 的浏览器，无需关心对象是用什么语言写的，也无须关心它是以DLL还是以另外的过程来执行的。从浏览器端看，无任何区别。这样一个通用的处理技巧非常有 用。例如，由用户协调运行的两个应用，可以将它们的共同作业部分作为COM对象间的交互来实现（当然，现在的OLE复合文档也能做到）。为在浏览器中执行 从Web服务器下载的代码，浏览器可把它看作是COM对象，也就是说，COM技术也是一种打包可下载代码的标准方法（ActiveX控件就是执行这种功能 的）。甚至连应用与本机OS进行交互的方法也可以用COM来指定，例如在Windows和Windows NT中用的是新API，多数是作为COM对象来定义的。可见，COM虽然起源于复合文档，但却可有效地适用于许多软件问题，它毕竟是处在底层的基础技术。 用一句话来说，COM是独立于语言的组件体系结构，可以让组件间相互通信。随着计算机网络的发展，COM进一步发展为分布式组件对象模型，这就是 DCOM，它类似于CORBA的ORB，本文对此将不再做进一步的阐述。通过上面的讲述相信大家一定对ActiveX、OLE和COM/DCOM的关系有 了一个清楚的了解。

使用Ｗｉｎｄｏｗｓ的人对于ＡｃｔｉｖｅＸ控制一定不会陌生，它提供了一种类似于ＤＬＬ动态链接库的调用，不过它与ＤＬＬ的唯一区别就是ＡｃｔｉｖｅＸ不 注册不能被系统识别并使用。那么，当我们得到一个ＡｃｔｉｖｅＸ没有被正确安装且不能使用的消息后，又要安装ＡｃｔｉｖｅＸ怎么办呢？１． Ｒｅｇｓｖｒ３２程序法在Ｗｉｎｄｏｗｓ的Ｓｙｓｔｅｍ文件夹下有一个ｒｅｇｓｖｒ３２．ｅｘｅ的程序，它就是Ｗｉｎｄｏｗｓ自己带的ＡｃｔｉｖｅＸ注册 和反注册工具。利用它也能够非常方便地注册ＡｃｉｔｖｅＸ控件，它的用法为：ｒｅｇｓｖｒ３２／ｕ／ｓ／ｎ／ｉ ｄｌｌｎａｍｅ， ｄｌｌｎａｍｅ其中ｄｌｌｎａｍｅ为ＡｃｔｉｖｅＸ控件文件名，建议在安装前拷贝到Ｓｙｓｔｅｍ文件夹下参数有如下意义：／ｕ － 反注册控件／ｓ － 不管注册成功与否，均不显示提示框／ｃ － 控制台输出／ｉ － 跳过控件的选项进行安装 （与注册不同）／ｎ － 不注册控件，此选项必须与／ｉ 选项一起使用例如笔者要注册一ａｍｏｖｉｅ．ｏｃｘ控件，则打入 ｒｅｇｓｖｒ３２ ａｍｏｖｉｅ．ｏｃｘ即可，要反注册它时只需使用 ｒｅｇｓｖｒ３２ ／ｕ ａｍｏｖｉｅ．ｏｃｘ就行了。２．注册表法所谓注册ＡｃｉｔｖｅＸ，无非是将一些信息记录在Ｗｉｎｄｏｗｓ的注册表中，如Ｓｈｏｃｋｗａｖｅ Ｆｌａｓｈ Ｏｂｊｅｃｔ控件，我们可以运行Ｒｅｇｅｄｉｔ．ｅｘｅ注册表编辑程序，利用关键字进行搜索，然后把搜索得到后的注册表导出为一ＲＥＧ注册表文件，再将其 相应的ＡｃｔｉｖｅＸ文件拷贝到Ｗｉｎｄｏｗｓ的Ｓｙｓｔｅｍ文件夹（一般ＡｃｔｉｖｅＸ的文件名为ＯＣＸ，安装在Ｗｉｎｄｏｗｓ的Ｓｙｓｔｅｍ文件夹 内）下，最后在要安装ＡｃｔｉｖｅＸ的机器上双击导入刚才导出的注册表文件即可完成安装。

Activex,OLE,COM都是微软的一些技术标准。Ole比较老后来发展成Activex，再后来发展成为COM OCX，DLL是扩展名。 Activex有两种扩展名OCX和DLL。实际上你可以把它们的扩暂名字调换。 COM作为ActiveX的更新技术，扩展名也有可能是DLL DLL文件还有可能是动态链接库。主要是装载一些函数，可以动态加载。 com的前景 以后一种比较理想的应用程序模式就是Ｗｅｂ化（条件是网络速度足够快），未来的软件应该不存在客户应用软件的说法了，客户就只有浏览器，浏览器就是操作系统，客户一边下载一边使用，当然下载的都是一个个功能独立的模块。而这些功能独立的模块就是com组件，一般的dll是不能这么用的。