关于获取COM对象和接口探讨

汤宏鹏 王杪（抚顺石油学院计算机系 113001）

摘要：在这篇文章中，我们讨论了如何创建对象，以及如何破坏，如何调用接口的方法程序和切换接口。同时也介绍了用于标识对象和接口的各种GUID的概念，还有所需的注册项，这样COM才能知道如何创建您的对象。

关键词：COM GUID 对象 标准接口 自定义接口

1 COM的标识符

我们需要一些标识符来表示COM世界中的各种实体。首先，对象类型(或称"类)需要一个标识符。其次，各接口也需要标识符。但是，我们应该用什么作为标识符呢？32位整数?64位整数?我们可以用它们作标识符。但是有一个问题：在所有的计算机上，标识符都必须是独一无二的，因为无法知道会在什么计算机上安装组件。对象和接口需要在所有的计算机上都使用相同的标识符，这样才能使任何客户程序都可以使用该组件。另外，不能有其他对象或接口使用这个标识符，不管其他对象或接口来自何处。也就是说，这些标识符必须是全球唯一的。

创建这种标识符的算法和数据格式是存在的。通过使用计算机的唯一网络卡ID、当前时间和其他数据，一个称为GUIDGEN.EXE的程序就可以创建这种标识符，称为GUID（全球唯一标识符）。GUID是按16字节（128位）结构存储的。

在C++中，COM头文件为GUID定义了数据类型，即CLSID（类标识符GUID）和IID（接口标识符GUID）。由于这些16字节结构有些大，因而不能按值传递，所以当传递GUID时，需要使用REFCLSID和REFIID作为参数的数据类型。您需要为每个对象类型创建一个CLSID，为每个自定义接口创建一个IID。

2 接口

2.1 标准接口

COM定义了大量的标准接口及其相关的IID。例如，所有接口的母辈IUnknown的IID是"00000000-0000-0000-c000-000000000046"（连字符是书写GUID的标准方式）。这个IID是由COM定义的，您永远也不必直接引用它：而应使用IID_IUnknown常量，这是在头文件中定义的。

IUnknown接口有三个函数：

```
HRESULT QueryInterface(REFIID riid,void**ppvObject);
ULONG AddRef();
ULONG Release();
```

稍后我们将详细讨论这些函数的功用。

使用标准接口可以完成大量COM编程工作—其中的大部分工作就是提供标准接口的实现细节，这样，其他的COM客户程序和对象就可以使用您的对象了。

2.2 自定义接口

自定义接口是您创建的接口。要为这些接口创建您自己的IID，并且定义您自己的函数。我们的IFoo接口就是一个自定义接口。通过运行我计算机上的GUID生成器，我已经定义了一个IID，称为IID_Ifoo（它的值是"13C0205C-A753-11dl-A52D-0000F8751BA7"）。

回忆一下，原来的类声明是：

```
class IFoo {
    virtual void Funcl(void)=0;
    virtual void Func2(int nCount)=0;
};
```

我们将略作修改，就将它变成COM兼容的：

```
Interface IFoo:IUnknown {
Virtual HRESULT STDMETHODCALLTYPE Funcl
(void) = 0;
virtual HRESULT STDMETHODCALLTYPE Func2(int
nCount)=0;
};
```

使用上面所说的宏，就变成：

```
Interface IFoo:IUnknown {
STDMETHOD Funcl(void) PURE;
STDMETHOD Func2(int nCount) PURE;
};
```

"Interface"并不是C++中的关键字，而是在相应的COM头文件中用#define定义为"stuct"的。(回忆一下，在C++中，类和结构是相同的，只是在默认情况下，结

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

构使用公共继承和访问，而不是私有的)。STDMETHOD使用STDMETHODCALLTYPE，它定义为__stdcall，这表明编译器要为这些函数生成标准的函数调用序列。我们使用这些宏是因为将我们的代码移植到不同的平台上时，它们的定义会改变。

3 对象的创建

一旦我们的CLSID与一个对象类型相关(将在稍后详细探讨)，就可以创建一个对象。正如所证明的，这很简单—只是一个函数的调用：

```
IFoo * pFoo=NULL;
HRESULT hr = CoCreateInstance (CLSID_Foo, NULL,
CLSCTX_ALL,IID_IFoo, (void**) &pFoo);
```

如果CoCreateInstance成功了，它会创建CLSID GUID CLSID_Foo标识的对象的一个实例。注意，没有“对象的指针”这回事；相反，我们总是通过一个接口指针引用对象。所以我们必须指定我们需要哪个接口(IID_IFoo)，并将一个指针传递到某处，以便让CoCreateInstance存储该接口指针。

一旦我们调用了一个函数，需要进行检查，以确保调用成功，并且接下来使用该对象：

```
if(SUCCEEDED(hr)){
    pFoo->Funcl(); // 调用方法程序。
    pFoo-> Func2(5);
    pFoo->Release(); // 当处理之后，必须释放接口。
}
else // 创建失败…
```

CoCreateInstance返回一个HRESULT，以表明它是否成功。因为非负数表示成功，我们总是使用SUCCEEDED宏来检查结果。实际上，最普通的成功代码S_OK是零，所以象“if(hr)// Success”这样的检查根本不起作用。一旦成功地创建了该对象，可以如上所示，使用接口指针来调用接口的方法程序。

当处理完接口指针后，需要通过调用Release来释放该接口指针，这是至关重要的。由于所有接口都是从IUnknown导出的，所以所有接口都支持Release。当您告诉COM对象您已经处理完它时，由它自己释放自己，但是它需要您告诉它您何时处理完。如果忘记了调用Release，该对象会被泄漏(并且被锁在内存中，至少要等到关闭应用程序，甚至要等到系统重新启动)。弄乱对象生命期是非常普遍的COM编程问题，而且难于发现。所以，要从现在开始小心谨慎。如果我们真正创建了某接口，就只能释放该接口。

图1是我们新创建对象的图例。按约定，IUnknown没有标识；它总是画在对象的右上角。所有其他接口画在左边。

图1第一个简单的COM对象，带有无标识IUnknown。

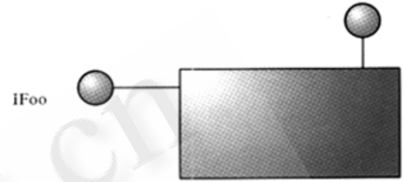

图 1

由于我们实现了IUnknown，所以就有了一个COM对象。(就像画一个连接器一样简单!)

如果将一个IFoo2接口添加到该对象，总共就有了三个接口，如图2所示。

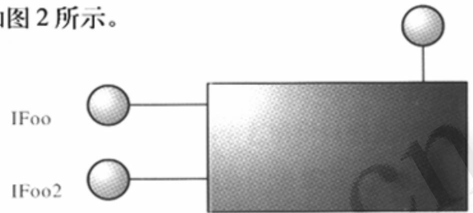

图 2

图2理论版2.0，支持IFoo和IFoo2。

4 GUID和注册表

那么COM是如何找到对象的代码，以便创建对象的呢?很简单：它在注册表中找。当安装了一个COM组件时，它必须在注册表中建立注册项。对于我们的Foo类，注册项可能会是这样：

```
HKEY_CLASSES_ROOT
  CLSID
   {E312522 E-A7B7-11D1-A52E-0000F8751BA7}="Foo Class"
InprocServer32="D://ATL Examples/Foo//Debug//Foo.dll"
```

大多数对象会有一些附加项，但是我们现在暂时忽略这些项。

在HKEY_CLASSES_ROOT/CLSID，有一个我们的类CLSID的注册项。这就是CoCreateInstance如何查找组件的DLL名称的。当您为CoCreateInstance提供了CLSID，它会找到DLL名称，加载这个DLL，并且创建该组件(稍

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

后将详细讨论)。

如果服务程序是线外的或远程的，该注册项会有所不同，但是重要的是这些信息存在，所以COM可以启动服务程序并且创建该对象。

如果知道对象的名称(ProgID)，但不知道它的CLSID，就可以在注册表中查找CLSID。对于我们的对象，有这样一个注册项:

```
HKEY_CLASSES_ROOT
  Foo.Foo="Foo Class"
     CURVER="Foo.Foo.1"
     CLSID="{E312522E-A7B7-11D1-A52E-0000F8751BA7}"
  Foo.Foo.1="Foo Class"
     CLSID="{E312522E-A7B7-11D1-A52E-0000F8751BA7}"
```

"Foo.Foo"是独立版本的ProgID,Foo.Foo.1是ProgID。如果您从Visual Basic中创建一个Foo对象，可使用其中一个ProgIDs查找CLSID。(注意，在当前版本，ATL向导还不能完全正确地创建注册项：它会漏掉上面所示的前两个CLSID关键字。不要忘记为独立版本的ProgID复制CLSID。)

5 模块、组件类和接口

一个模块(DLL或EXE)有可能(实际上是通常)实现多个COM组件类。如果是这样，会有多个CLSID注册项引用相同的模块。

这样我们现在可以定义模块、类和接口之间的关系。一个模块(您连编和安装的基本单元)可以实现一个或多个组件，每个组件在注册表中都有自己的CLSID和指向模块的文件名的注册项，并且每个组件至少实现两个接口：IUnknown和一个提供组件功能的接口。图3表明了这点。

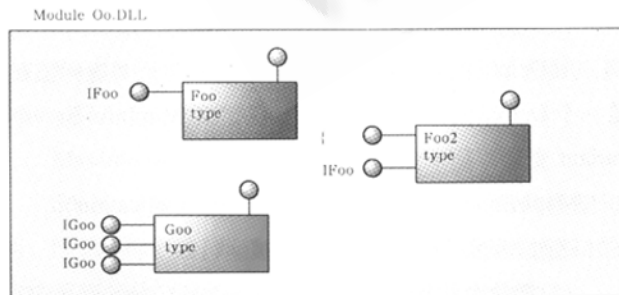

图 3

图3模块Oo.DLL包含三个对象(Foo、Goo和Hoo)的实现细节。每个对象实现了IUnknown和一个或多个附加接口。

可以说，我们有了一个新的改进了的Foo2对象，它实现了两个自定义接口：IFoo和IFoo2。我们已经知道了如何使用CoCreateInstance创建这样一个对象，也知道了如何获得一个指向三个接口(不要忘记IUnknown)之一的指针。

当我们得到这样的接口指针之后，怎样才能得到该对象的其他接口的接口指针呢？不能再调用CoCreateInstance—这会创建一个新对象，这并不是我们希望的，我们只需要现有对象的另一个接口。

这就需要IUnknown::QueryInterface来解决了。记住，由于所有的接口都从IUnknown继承，所以它们都执行QueryInterface。这样，我们只需使用第一个接口指针来调用QueryInterface，以便得到第二个接口指针:

```
IFoo*pFoo=NULL;
HRESULT hr = CoCreateInstance(CLSID_Foo2, NULL,
CLSCTX_ALL,IDD_IFoo, (void**) &pFoo);
if(SUCCEEDED(hr)) {
  pFoo->Funcl(); // 调用 IFoo::Funcl
  IFoo2*pFoo2=NULL;
  hr = pFoo - >QueryInterface(IID_IFoo2, (void**)
&pFoo2);
  if(SUCCEEDED(hr)) {
    int inoutval=5;
    pFoo2 - >Func3(&inoutval); //IFoo2::Func3
    pFoo2 - >Release();
  }
pFoo->Release();
}
```

我们向QueryInterface传递了所需接口的IID，还有一个指向QueryInterface保存新接口指针的位置的指针。一旦QueryInterface返回成功，我们就可以使用该接口指针来调用该接口的函数。

一定要注意，当我们处理完两个接口指针之后必须释放这两个指针。如果释放其中的一个指针失败，就会泄漏该对象。由于我们只能通过接口指针引用该对象，所以必须释放每个得到的接口指针，这样才能作为一个整体释放该对象。

IUnknown有其他两个函数：AddRef和Release。我们已经看到，使用Release告诉一个对象，已经处理完一个接口指针。那么您什么时候使用AddRef呢?

(下转第42页)

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

(上接第45页)

6 引用计数

大多数COM对象都保留一个引用计数(reference count)—也就是说，它们需要跟踪有多少个该对象的接口指针正在使用。如果所有对象接口的引用计数变成零时，该对象就可以释放。我们不必明确地释放该对象；只需释放对象的所有接口指针，对象会在合适的时候释放自己。

AddRef增加引用计数，而Release减少引用计数。所以，如果没有调用AddRef，为什么必须调用Release呢？

每当QueryInterface为一个对象分配一个新指针时，QueryInterface有责任在返回该指针前调用AddRef。这就是为什么不必为得到的指针调用AddRef。QueryInterface为我们做了。(注意，CoCreateInstance调用QueryInterface，而QueryInterface调用AddRef，所以对象的第一个接口指针也是这样)。

对于调用了AddRef的相同接口指针，也需要调用Release。如果对象需要，它们可以一个接口一个接口地跟踪引用。上面的代码小心地做到了这一点，对应Release调用的正确配对的隐含AddRef调用—每个接口指针一个Release调用。

如果复制了一个接口指针，则需要调用AddRef，这样该接口的引用计数才准确。至少何时需要何时不需要有点复制，但是各种COM参考书都有详细的介绍。

各种巧妙的指针类使得处理IUnknown变得容易多了(实际上是自动的)。在ATL和Visual C++5.0中有几个这样的类。如果您使用另一种语言，例如Visual Basic或Java，该语言对COM的实现会正确地处理Iunknown方法程序。■

参考文献

1 *Dale Rogerson, Inside COM (Microsoft Press, 1997).*

2 *Don Box, Essential COM (Addison-Wesley, 1998).*

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn