COM实用入门教程　　第二讲

主讲人：阙海忠

VC知识库网站 (www.vckbase.com) 拍摄制作

本讲要点：

- 一、内存资源何时释放；
- 二、引用计数的原理；
- 三、AddRef与Release的实现与使用；
- 四、引用计数的优化。

内存资源何时释放

在上一节的例子中，我们用到了new CA()与delete pA。我们知道我们创建了一个组件，最终在不用这个组件的时候，是应该把它销毁了。不过，什么时候才是“不用这个组件的时候”呢？

假设上例中，我们可能会把pIUnknown传给CB类的成员变量。如：

```
if(...)                    //可能为真，也可能为假
{
        CB *pB = new CB(pIUnknown);
}
```

我们怎么知道何时我们不会再用到这个pA所指向的组件？当然，你可能会回答“在主函数的最后面执行delete pA，因为那时pA一定不用了。”

内存资源何时释放

在主函数的最后面执行delete pA确实是一个可行的办法。但却不是好办法。因为这样子最终是释放了pA的内存资源，不过却不是“及时”(在pA所指的组件不用时)地释放内存资源。如果一个程序，所有的资源在不用时都没有及时释放，这个程序在运行中所占用的内存将是巨大的。如何解决这个问题呢？这就需要引用计数技术。

这一讲，主要讲解如下要点：

- 一、内存资源何时释放；
- 二、引用计数的原理；
- 三、AddRef与Release的实现与使用；
- 四、引用计数的优化。

引用计数的原理

- 引用计数技术就是用来管理对象生命期的一种技术。
- 对象O可能同时被外界A，外界B，外界C引用。也就是说外界A，外界B，外界C可能都在使用对象O。
- 每次当对象被外界引用时，计数器就自增1。
- 每次当外界不用对象时，计数器就自减1。

引用计数的原理

- 在计数值为零时，对象本身执行delete this,销毁自己的资源。
- 引用计数使得对象通过计数能够知道何时对象不再被使用，然后及时地删除自身所占的内存资源。
- IUnknown接口的AddRef与Release就是引用计数的实现方法。

这一讲，主要讲解如下要点：

- 一、内存资源何时释放；
- 二、引用计数的原理；
- 三、AddRef与Release的实现与使用；
- 四、引用计数的优化。

AddRef与Release的实现

- 查看Section2Demo1关于AddRef与Release的实现。

AddRef的意义与使用

- AddRef，使组件的引用计数自增1。
- 在返回之前调用AddRef。比如组件的构造函数，QueryInterface函数。
- 在赋值之后调用AddRef。比如pIX3 = pIX2，这时需要pIX3->AddRef();

Release的意义与使用

- Release，使组件引用计数自减1，如果引用计数为零，释放本身的内存资源。
- 在接口使用完之后，调用Release。
- 查看Section2Demo1，关于AddRef与Release的使用，并适当注释一些AddRef或Release查看CA的析构函数是否运行或会不会出现对无效指针(野指针)的操作。

这一讲，主要讲解如下要点：

- 一、内存资源何时释放；
- 二、引用计数的原理；
- 三、AddRef与Release的实现与使用；
- 四、引用计数的优化。

引用计数的优化

- 刚才的例子中，我们看到了

```
if (SUCCEEDED(hr))
{
        IX *pIX3 = NULL;
        pIX3 = pIX2;
        pIX3->AddRef();

        pIX3->Fx1();
        pIX3->Fx2();

        pIX3->Release();
        pIX3 = NULL;
   }
```

引用计数的优化

- 对于pIX2与pIX3来说，都是同一个接口，生命期是一样的，这个接口在一个块({})中，执行了一次的AddRef，一次的Release，其实就相当于没有执行AddRef,与Release的效果。是否可以优化为下一页的代码呢？

引用计数的优化

```
if (SUCCEEDED(hr))
   {
         IX *pIX3 = NULL;
         pIX3 = pIX2;

         pIX3->Fx1();
         pIX3->Fx2();
   }
```

引用计数的优化

- 这种优化可行吗？答案是可行的！因为这种优化符合了引用计数优化的“局部变量原则”
- 引用计数的优化原则：

一、输入参数原则：

- 输入参数指的是给函数传递某个值的参数。在函数体中将会使用这个值但却不会修改它或将其返回给调用者。在C++中，输入参数实际上就是那些按值传递的参数。
- 对传入函数的接口指针，无需调用AddRef与Release

二、局部变量原则

对于局部复制的接口指针，由于它们只是在函数的生命期内才存在，因此无需调用AddRef与Release

引用计数的优化

- 输入参数原则：

```
void Fun(IX *pIXParam)        //参数传递存在赋值过程
{
        //pIXParam->AddRef();  //可优化，注释掉
        pIXParam->Fx1();
        pIXParam->Fx2();
        //pIXParam->Release();//可优化，注释掉
}
```

引用计数的优化

- 局部变量原则：

```
void Fun(IX *pIX)
{
        IX *pIX2 = pIX;
        //pIX2->AddRef();       //可优化，注释掉
        pIX2->Fx1();
        pIX2->Fx2();
        //pIX2->Release();      //可优化，注释掉
}
```

引用计数的优化

- 以下代码可以优化吗？

```
void  Fun(IX **ppIX)
{
        (*ppIX)->Fx1();
        (*ppIX)->Fx2();
        (*ppIX)->Release();          //可以优化吗？
        *ppIX = m_pIXOther;
        (*ppIX)->AddRef();           //可以优化吗？
        (*ppIX)->Fx1();
        (*ppIX)->Fx2();
}
```

引用计数的优化

- 答案是否定的！因为它不是输入参数原则，而是输入-输出参数原则。此原则下，引用计数不能优化！

//以上两句务必要运行，因为*ppIX 与m_pIXOther不一个属性同一个组件。

//比如假设*ppIX是指向第一次的new CA()，而m_pIXOther却是指向第二次的new CA()。

//或者*ppIX是指向new CA()，而m_pIXOther是指向new CZ()，CA与CZ的共同点，只是都继承了IX接口而已。

引用计数的优化

- 引用计数，带来了高效的内存资源管理方法，能及时地释放不再使用的资源。但却带来了编码的麻烦。在后续的讲解中，会讲到对引用计数的封装，也就是智能指针，到时组件的客户不再编写AddRef与Release代码，也不需要编写delete代码，便可以方便，舒心地进行内存资源的管理。

回顾

- 这一讲，主要讲解如下要点：

一、内存资源何时释放；
二、引用计数的原理；
三、AddRef与Release的实现与使用；
四、引用计数的优化。