Implementing two-phase initialization with ATL

devblogs.microsoft.com/oldnewthing/20240118-00

January 18, 2024

Raymond Chen

In an attempt to solve problems with exceptions thrown out of constructors that hand out COM references in your class implement with ATL, you might notice this nice extension point called `FinalConstruct()` and use it for the second phase of two-phase construction.

```
// ATL - this code is wrong
class MyPage : public CComObjectRootEx<CComMultiThreadModel>,
    public CComCoClass<MyPage>,
    public IPage
{
public:
    DECLARE_PROTECT_FINAL_CONSTRUCT()

    HRESULT FinalConstruct() try
    {
        Application::LoadComponent(this, blah, blah);
        something_that_might_throw();
        return S_OK;
    }
    CATCH_RETURN();

    ⟦ ... ⟧
};
```

You thought you were clever and remembered that ATL runs the constructor with a reference count of zero, so you deferred the operations that use COM references to the `Final-Construct()`, and you used `DECLARE_PROTECT_FINAL_CONSTRUCT()` to ensure that `Final-Construct()` runs with a nonzero reference count.

However, if you look at how `CComCoClass::CreateInstance` uses `FinalConstruct()`, you'll see that it doesn't really work for two-phase construction:

```
template<class Base>
/* static */
HRESULT WINAPI CComObject<Base>::CreateInstance(
    CComObject<Base>** pp) throw()
{
    ATLASSERT(*pp == NULL);
    if (pp == NULL)
        return E_POINTER;
    *pp = NULL;

    HRESULT hRes = E_OUTOFMEMORY;
    CComObject<Base>* p = NULL;
    ATLTRY(p = _ATL_NEW CComObject<Base>())
    if (p != NULL)
    {
        p->SetVoid(NULL);
        p->InternalFinalConstructAddRef();
        hRes = p->FinalConstruct();
        p->InternalFinalConstructRelease();
        if (hRes != S_OK) {
            delete p;
            p = NULL;
        }
    }
    *pp = p;
    return hRes;
}
```

Observe that if `FinalConstruct()` fails, the object is outright `delete`d; any `AddRef` that occurred during `FinalConstruct()` won't prevent the object's destruction.

You will have to implement the two-phase construction manually.

```
class MyPage : public CComObjectRootEx<CComMultiThreadModel>,
    public CComCoClass<MyPage>,
    public IPage
{
public:
    HRESULT InitializeComponent() noexcept try
    {
        Application::LoadComponent(this, blah, blah);
        something_that_might_throw();
        return S_OK;
    }
    CATCH_RETURN();

    ⟦ ... ⟧
};

HRESULT CreateMyPage(CComObject<MyPage>** result)
{
    *result = NULL;

    CComObject<MyPage>* page;

    HRESULT hr = CComObject<MyPage>::CreateInstance(&page);
    if (FAILED(hr)) return hr;

    CComPtr<CComObject<MyPage>> pageRef(page);

    hr = page->InitializeComponent());
    if (FAILED(hr)) return hr;

    *result = pageRef.Detach();
    return S_OK;
}
```

The important things to note are

- We bump the reference count from 0 to 1 (by putting it in a `CComPtr`) before calling `InitializeComponent()`, so that the COM references we hand out have a nonzero reference count.
- We use a `CComPtr` so that the reference will be released automatically if `Initialize-Component()` throws an exception or returns a COM failure.
- The `CComPtr` destructor does a `Release()` rather than a `delete`, so any extra references created by `InitializeComponent()` are honored.