The skeleton marshaler that does default marshaling

devblogs.microsoft.com/oldnewthing/20220616-00

June 16, 2022

Raymond Chen

Last time, we took an initial look at the mechanics of how COM marshaling is performed. Today, we'll build a custom marshaler that provides no custom behavior. This sounds silly, but it's useful because we can use it as a starting point for custom behavior.

```
struct MyClass : public IMarshal /* ... and other interfaces */
{
    // QueryInterface, AddRef, and Release left as an exercise

    IUnknown* CastToUnknown() { return static_cast<IMarshal*>(this); }

    STDMETHODIMP GetUnmarshalClass(
        REFIID riid, void* pv, DWORD dwDestContext,
        void* pvDestContext, DWORD mshlflags,
        CLSID *clsid)
    {
        ComPtr<IMarshal> marshal;
        RETURN_IF_FAILED(CoGetStandardMarshal(riid, CastToUnknown(), dwDestContext,
                                              pvDestContext, mshlflags, &marshal));
        RETURN_IF_FAILED(marshal->GetUnmarshalClass(riid, pv, dwDestContext,
                                              pvDestContext, mshlflags, clsid));
        return S_OK;
    }

    STDMETHODIMP GetMarshalSizeMax(
        REFIID riid, void* pv, DWORD dwDestContext,
        void* pvDestContext, DWORD mshlflags,
        LPDWORD size)
    {
        ComPtr<IMarshal> marshal;
        RETURN_IF_FAILED(CoGetStandardMarshal(riid, CastToUnknown(), dwDestContext,
                                              pvDestContext, mshlflags, &marshal));
        RETURN_IF_FAILED(marshal->GetMarshalSizeMax(riid, pv, dwDestContext,
                                              pvDestContext, mshlflags, size));
        return S_OK;
    }

    STDMETHODIMP MarshalInterface(
        IStream* pstm,
        REFIID riid, void* pv, DWORD dwDestContext,
        void* pvDestContext, DWORD mshlflags)
    {
        ComPtr<IMarshal> marshal;
        RETURN_IF_FAILED(CoGetStandardMarshal(riid, CastToUnknown(), dwDestContext,
                                              pvDestContext, mshlflags, &marshal));
        RETURN_IF_FAILED(marshal->MarshalInterface(pstm, riid, pv, dwDestContext,
                                              pvDestContext, mshlflags));
        return S_OK;
    }
```

The interfaces on the marshaling side ask the system, "Hey, what marshaler would you have used for this object if it didn't do custom marshaling?"[1] The `MyClass` object will probably implement interfaces beyond `IMarshal`, so we can't just pass `this` because that will result

in an ambiguous conversion. We provide a custom `CastToUnknown()` method that picks one of the `IUnknown*` base classes arbitrarily. If you're using WRL, then the `CastToUnknown()` method has been provided for you; otherwise, you get to write one of your own.[2]

The methods on the unmarshaling and cleanup side are even simpler: Your unmarshaling and cleanup functions will never be called, because the unmarshaling and cleanup are performed by the unmarshal class, which we delegated to the standard marshaler. Therefore, you can just return "Huh? Shouldn't ever get here."

```
    STDMETHODIMP UnmarshalInterface(IStream* pstm, REFIID riid, void** ppv)
    {
        *ppv = nullptr;
        return E_UNEXPECTED;
    }

    STDMETHODIMP ReleaseMarshalData(IStream* pstm)
    {
        return E_UNEXPECTED;
    }

    STDMETHODIMP DisconnectObject(DWORD dwReserved)
    {
        return E_UNEXPECTED;
    }
};
```

Next time, we'll take this skeleton and use it to implement marshal-by-value.

[1] The `CoGetStandardMarshal` function probes the provided object's unmarshal class, which creates recursion because it will call `GetUnmarshalClass` again, which we delegate back out to the standard marshaler. Fortunately, the system has a recursion detector and figures that if the call to the `CoGetStandardMarshal` function, then the object's `IMarshal` is just trying to delegate to the standard one.

[2] You are probably not going to implement any interfaces that derive from `IMarshal`, so the version provided here will probably work fine: The cast to `IMarshal*` is almost certainly unambiguous. However, I hid it inside a call to `CastToUnknown` so that it won't look like I'm saying "Oh, the second parameter to `CoGetStandardMarshal` must be an `IMarshal*`. It doesn't. It just needs to be an `IUnknown*`.

Raymond Chen

Follow