The C++/WinRT query_interface_tearoff extension point, and using it for COM aggregation

devblogs.microsoft.com/oldnewthing/20210913-00

September 13, 2021

Raymond Chen

The C++/WinRT library's `implements` template does the heavy lifting of implementing COM classes. One of the extension points is a method called `query_interface_tearoff`. This method is called as part of the implementation of the `IUnknown::QueryInterface` method if the caller is asking for an interface that wasn't declared as part of the `implements` type parameter list. This gives you a chance to support additional interfaces.

From its name, it's apparent that the primary use case for this is COM tear-off interfaces. But you can use it for any scenario where you want to support an interface that you didn't list in your `implements`. For example, the interface might be dynamically-generated. But today we're going to use it for COM aggregation.

Let's say that we want to aggregate the free-threaded marshaler. Yes, I know that C++/WinRT does this automatically, but let's do it manually just to show how it's done.

```
struct MyFreeThreaded :
    winrt::implements<MyFreeThreaded, ::IAgileObject, winrt::non_agile>
{
  winrt::IUnknown m_ftm;

  MyFreeThreaded()
  {
    winrt::check_hresult(CoCreateFreeThreadedMarshaler(this, m_ftm.put()));
  }

  int32_t query_interface_tearoff(winrt::guid const& id, void** object)
        const noexcept override
  {
    if (id == winrt::guid_of<::IMarshal>()) {
      return m_ftm.as(id, object);
    }
    return E_NOINTERFACE;
  }
};
```

The `MyFreeThreaded` object implements `IAgileObject`, which is a marker interface for free-threaded objects. It also uses the `winrt::non_agile` marker to tell C++/WinRT not to implement free-threading support for this object. Because we're about to do it manually!

We declare a member variable `m_ftm` which holds the free-threaded marshaler. At construction we create the free-threaded marshaler passing ourselves as the controlling unknown. This makes the free-threaded marshaler act as if it were part of our object, and it will forward all interface requests back to the main object. However, the `IUnknown` produced by `CoCreateFreeThreadedMarshaler` is a special one that does not delegate its `IUnknown::QueryInterface` method. This lets you access the interfaces of the aggregated object. (Without it, any attempt to obtain an interface from the aggregated object would just be forwarded back to the outer object.)

To complete the circle, we forward requests for `IMarshal` into the free-threaded marshaler by overriding `query_interface_tearoff`. That way, if somebody asks for `IMarshal` — and by *somebody* I mean *COM itself*—we forward the request into the aggregated object, which says "Free-threaded marshaler at your service!"

An important detail of implementing your own `query_interface_tearoff` is that you should return `E_NOINTERFACE` (or forward to the base class) if you don't handle the interface.

That's a pretty quick rundown of doing COM aggregation in C++/WinRT. It'll come in handy in a few months.

Raymond Chen

Follow