Reducing chattiness by querying for multiple interfaces at once, part 2

devblogs.microsoft.com/oldnewthing/20220316-00

March 16, 2022

Raymond Chen

Last time, we saw how we can use `MULTI_QI` to get multiple interfaces packed into a single server call. This saves us from having to issue separate `QueryInterface` calls, avoiding a round-trip call to the server for each interface.

It was relatively easy for us to do this in our sample because all of the `QueryInterface` calls were in the same function. But what if they are spread out?

```
// Error checking elided for expository purposes.

void Gadget::DoSomethingWithWidget()
{
    // Create a widget.
    wil::com_ptr<IWidget> widget;
    CoCreateInstance(CLSID_Widget, nullptr,
        CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&widget));

    LoadWidgetWithSite(widget.get(), site, m_fileName);

    // Ready to do widget things.
    widget->DoSomething();
}

void LoadWidgetWithSite(IWidget* widget,
    IUnknown* site, PCWSTR fileName)
{
    // Set the site.
    wil::com_ptr<IObjectWithSite> objectWithSite;
    widget->QueryInterface(IID_PPV_ARGS(&objectWithSite));
    objectWithSite->SetSite(site);

    // Load it from the file.
    wil::com_ptr<IPersistFile> persistFile;
    widget->QueryInterface(IID_PPV_ARGS(&persistFile));
    persistFile->Load(fileName, STGM_READ);
}
```

If we want to take advantage of `CoCreateInstanceEx` and `MULTI_QI`, it looks like we'll have to pass those pre-queried interfaces to `LoadWidgetWithSite`, which makes it more unwieldy:

```
// Error checking elided for expository purposes.

void Gadget::DoSomethingWithWidget()
{
    // Create a widget and prefetch the interfaces.
    MULTI_QI mqi[3] = {
        { &__uuidof(IWidget), nullptr, 0 },
        { &__uuidof(IObjectWithSite), nullptr, 0 },
        { &__uuidof(IPersistFile), nullptr, 0 },
    };
    HRESULT hr = CoCreateInstanceEx(
        CLSID_Widget, nullptr, CLSCTX_LOCAL_SERVER,
        nullptr, 3, mqi);

    wil::com_ptr<IWidget> widget;
    widget.attach(mqi[0].pItf);

    wil::com_ptr<IObjectWithSite> objectWithSite;
    objectWithSite.attach(mqi[1].pItf);

    wil::com_ptr<IPersistFile> persistFile;
    persistFile.attach(mqi[2].pItf);

    if (hr != S_OK) {
        // Failed to get at least one interface.
        return;
    }

    LoadWidgetWithSite(widget.get(),
        objectWithSite.get(), site,
        persistFile.get(), m_fileName);

    // Ready to do widget things.
    widget->DoSomething();
}

void LoadWidgetWithSite(IWidget* widget,
    IObjectWithSite* objectWithSite, IUnknown* site,
    IPersistFile* persistFile, PCWSTR fileName)
{
    // Set the site.
    objectWithSite->SetSite(site);

    // Load it from the file.
    persistFile->Load(fileName, STGM_READ);
}
```

But it turns out that you don't have to rewrite all of your methods. All you have to do is prefetch the interfaces and *throw them away*!

```
// Error checking elided for expository purposes.

void Gadget::DoSomethingWithWidget()
{
    // Create a widget and prefetch the interfaces.
    MULTI_QI mqi[3] = {
        { &__uuidof(IWidget), nullptr, 0 },
        { &__uuidof(IObjectWithSite), nullptr, 0 },
        { &__uuidof(IPersistFile), nullptr, 0 },
    };
    HRESULT hr = CoCreateInstanceEx(
        CLSID_Widget, nullptr, CLSCTX_LOCAL_SERVER,
        nullptr, 3, mqi);

    wil::com_ptr<IWidget> widget;
    widget.attach(mqi[0].pItf);

    if (mqi[1].pItf) mqi[1].pItf->Release();
    if (mqi[2].pItf) mqi[2].pItf->Release();

    if (hr != S_OK) {
        // Failed to get at least one interface.
        return;
    }

    // The rest is the same as the non-MULTI_QI version.
    LoadWidgetWithSite(widget.get(), site, m_fileName);

    // Ready to do widget things.
    widget->DoSomething();
}

void LoadWidgetWithSite(IWidget* widget,
    IUnknown* site, PCWSTR fileName)
{
    // Set the site.
    wil::com_ptr<IObjectWithSite> objectWithSite;
    widget->QueryInterface(IID_PPV_ARGS(&objectWithSite));
    objectWithSite->SetSite(site);

    // Load it from the file.
    wil::com_ptr<IPersistFile> persistFile;
    widget->QueryInterface(IID_PPV_ARGS(&persistFile));
    persistFile->Load(fileName, STGM_READ);
}
```

Even though we threw the prefetched interfaces away, they have been cached in the proxy, and future calls to `QueryInterface` will return the cached value instead of sending a call all the way back out to the server.

The proxy also caches negative results, so if we had an optional interface, the proxy will remember that a query for that interface failed in the past, so when you ask for it again, it will return the error from the earlier `QueryInterface` without going to the server.

The rules for `IUnknown` regarding interface stability ensure that it is valid to cache the results of earlier `QueryInterface` calls.[1]

Even the `Release` calls on the interfaces won't result in a call out to the server: Only the final `Release` of a proxy results in a call to the server, and we still have an active reference in `widget`. (We took advantage of this behavior a little while ago.)

Mind you, the `MULTI_QI` structure is rather awkward to manage. Maybe we can use some C++ magic to make it easier. We'll look at that next time.

[1] You might say that `QueryInterface` "collapses the wave function" for interface detection. If your object is never asked "Do you support `IWidget`?" then it can exist in a quantum superposition state of "supports `IWidget`/doesn't support `IWidget`." But once somebody asks, the object must decide which way it wants to be, and has to stick with that decision for the remainder of its lifetime.

You can take advantage of the "quantum superposition state" in your objects. For example, your `Tool` object that might be an `IWidget`, or it might be an `IGadget`, depending on how it is configured. The client can reconfigure the Tool all it wants, but once it asks "Did I make a Widget?", the configuration is locked in. In practice, you see this pattern in the cases where the configuration is done by something like `IPersistStream`.

```
wil::com_ptr<IPersistStream> persist;
CoCreateInstance(CLSID_Tool, ..., IID_PPV_ARGS(&persist));

// The tool doesn't know what it is yet.

persist->Load(stream);

// The tool can delay the decision until somebody finally asks,
// "Are you a Widget?"

wil::com_ptr<IWidget> widget;
persist->QueryInterface(IID_PPV_ARGS(&widget));
```

Raymond Chen

Follow