Reducing chattiness by querying for multiple interfaces at once, part 1

devblogs.microsoft.com/oldnewthing/20220315-00

March 15, 2022

Raymond Chen

During performance analysis, you may discover that your usage of a remote COM object is too chatty, meaning that too much time spent communicating back and forth at the expense of actual work. We saw some time ago how you can marshal a buffer by value instead of by reference, so that there is only one round trip to the server to get the buffer.

But maybe your chattiness problem is with the `QueryInterface` calls:

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

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

// Set ourselves as its site.
wil::com_ptr<IObjectWithSite> objectWithSite;
widget->QueryInterface(IID_PPV_ARGS(&objectWithSite));
objectWithSite->SetSite(this);

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

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

Let's count the number of calls to the server, and how many of them are performing actual work.

Call	Purpose	Nature
`CoCreateInstance`	Create the widget	Real work

`QueryInterface`	Get the `IObjectWithSite`	Bookkeeping
`SetSite`	Set the site	Configuration
`QueryInterface`	Get the `IPersistFile`	Bookkeeping
`Load`	Load the widget	Initialization
`DoSomething`	Do something	Activity

There are six calls to the server, and a third of them are just bookkeeping.

We can batch together the `QueryInterface` by using `IMultiQI`:

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

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

// Get two interfaces with one call.
wil::com_ptr<IMultiQI> multiQI;
widget->QueryInterface(IID_PPV_ARGS(&multiQI));

MULTI_QI mqi[2] = {
    { &__uuidof(IObjectWithSite), nullptr, 0 },
    { &__uuidof(IPersistFile), nullptr, 0 },
};
HRESULT hr = multiQI->QueryMultipleInterfaces(2, mqi);

wil::com_ptr<IObjectWithSite> objectWithSite;
objectWithSite.attach(mqi[0].pItf);

wil::com_ptr<IPersistFile> persistFile;
persistFile.attach(mqi[1].pItf);

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

// Set ourselves as its site.
objectWithSite->SetSite(this);

// Load it from a file.
persistFile->Load(fileName, STGM_READ);

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

We were able to combine the two `QueryInterface` calls into one by issuing a batch query. Note that the `QueryInterface` for `IMultiQI` is not a server call: The `IMultiQI` interface is implemented locally on the proxy.

But wait, we can do even better: We can use `CoCreateInstanceEx` to obtain all thread interfaces as part of the initial creation:

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

// Create a widget and request three 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;
}

// Set ourselves as its site.
objectWithSite->SetSite(this);

// Load it from a file.
persistFile->Load(fileName, STGM_READ);

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

Now we have gotten rid of all of the bookkeeping calls.

Call	Purpose	Nature
`CoCreateInstanceEx`	Create the widget and get interfaces	Real work

`SetSite`	Set the site	Configuration
`Load`	Load the widget	Initialization
`DoSomething`	Do something	Activity

Okay, so now we get to talk about error checking.

There are three classes of results related to whether the interfaces could be obtained.

Interfaces obtained	QueryMultipleInterfaces	CoCreateInstanceEx
All	`S_OK`	
Some but not all	`S_FALSE`	`CO_S_NOTALL-INTERFACES`
None	Error	

Note that for `CoCreateInstanceEx`, there are other errors possible to report any problems creating the object, but I'm looking at the interface-related errors.

In our case, we need all of the interfaces, so anything that isn't `S_OK` is bad news, and we give up immediately.

There may be other cases where you are probing for an interface and will take advantage of it if present, but its absence should not be considered a failure. In that case, you would dig into the `MULTI_QI` to find out which interfaces could be obtained and which failed. You can use `SUCCEEDED(hr)` as a shortcut to detect that *something* was obtained.

Note that in our sample code above, the obtained interfaces are immediately transferred to smart pointers so that they will be released properly, even in the case where not all interfaces were obtained.

Now, it may be that the various calls to `QueryInterface` are scattered through the code, and it is unwieldy to query them at creation and then pass them around to all the places that query for them. We'll look at that case next time.

Bonus chatter: Note that the batched interface query is a significant improvement only for remote objects. For local objects, calls to the object occur directly, so there's no marshaling overhead. Furthermore, local objects are unlikely to support the `IMultiQI` interface at all.

Raymond Chen

Follow