Writing a marshal-by-value marshaler, part 2

devblogs.microsoft.com/oldnewthing/20220623-00

June 23, 2022

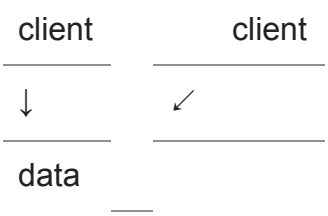
Raymond Chen

When we last left our marshal-by-value marshaler, we had it marshal by value for all cases that didn't leave the computer. But it turns out that this is actually more work than necessary: If the object is immutable (as marshal-by-value objects generally are) and is staying inside the same process, then we can just use the free-threaded marshaler, assuming the immutable state does not have thread affinity.

The idea here is that immutable objects are pretty much free-threaded already, seeing as they have no mutable state that requires synchronization. So there's no need to create a copy when marshaling between apartments within the same process; we can just have all the apartments access the object directly.

Old and busted:

New hotness:



Clients within the same process don't need a separate copy of the object. They can just share the object.

Of course, clients from other processes can take advantage of having their own copy, since that would avoid inter-process calls to access the data. However, we permit this only for other processes on the same computer, because processes on other computers may not have our custom marshaler installed.

This means that we want to treat each of the three categories differently:

Category	Desired marshaler
Same-process	Free-threaded marshaler
Same-machine	Marshal-by-value marshaler
Cross-machine	Standard (marshal-by-reference) marshaler

It sounds like we're going to have three cases to deal with, but we can collapse it down to two by realizing that the free-threaded marshaler falls back to the standard marshaler when the destination context is cross-machine.

```
bool ShouldMarshalByValue(DWORD dwDestContext)
{
    return // dwDestContext == MSHCTX_CROSSCTX || dwDestContext == MSHCTX_INPROC ||
           dwDestContext == MSHCTX_LOCAL || dwDestContext == MSHCTX_NOSHAREDMEM;
}
```

We no longer marshal by value in the same-process case, letting the free-threaded marshaler take care of that and the cross-machine case.

None of our marshal-by-value business logic needs to change. What changes is our fallback marshaler. Instead of falling back to the standard marshaler, we fall back to the free-threaded marshaler.

```
    STDMETHODIMP GetUnmarshalClass(
        REFIID riid, void* pv, DWORD dwDestContext,
        void* pvDestContext, DWORD mshlflags,
        CLSID *clsid)
    {
        if (ShouldMarshalByValue(dwDestContext)) {
            *clsid = CLSID_MyClass;
            return S_OK;
        }

        ComPtr<IMarshal> marshal;
        RETURN_IF_FAILED(CoCreateFreeThreadedMarshaler(nullptr, &marshal));
        RETURN_IF_FAILED(marshal->GetUnmarshalClass(riid, pv, dwDestContext,
                                                    pvDestContext, mshlflags, clsid));
        return S_OK;
    }
```

Identical one-line changes apply to `GetMarshalSizeMax` and `MarshalInterface`; I won't write them out.

Next time, we'll apply all that we learned to diagnosing a reference counting bug related to marshaling.

Raymond Chen

Follow