Inside C++/WinRT: Apartment switching: COM without COM

devblogs.microsoft.com/oldnewthing/20230130-00

January 30, 2023

Raymond Chen

One feature of the C++/WinRT library is that it can operate without the Component Object Model (COM) services. It still uses the COM application binary interface (ABI), but it doesn't rely on the system COM services. If they aren't available, it will just make do on its own.

This pattern of using COM as an ABI without using any COM services is sometimes called nano-COM. Some other Windows components follow this pattern, such as DirectX and Media Foundation.

In the case where COM is not active, C++/WinRT acts as if all threads are running in the multi-threaded apartment (MTA), and therefore all apartment-switching operations have no effect.

We have to add fallback code to the places where we ask COM for information or ask it to do something for us.

```
inline std::pair<int32_t, int32_t> get_apartment_type() noexcept
{
    int32_t aptType;
    int32_t aptTypeQualifier;
    if (0 == WINRT_IMPL_CoGetApartmentType(
        &aptType, &aptTypeQualifier))
    {
        return { aptType, aptTypeQualifier };
    }
    else
    {
        return { 1 /* APTTYPE_MTA */,
                 1 /* APTTYPEQUALIFIER_IMPLICIT_MTA */ };
    }
}
```

If we can't get the apartment type, then we must be running in nano-COM, and we report that we are running in the implicit MTA.

```
struct resume_apartment_context
{
    resume_apartment_context() = default;
    resume_apartment_context(std::nullptr_t) :
        m_context(nullptr), m_context_type(-1) {}
    resume_apartment_context(
        resume_apartment_context const&) = default;
    resume_apartment_context(
    resume_apartment_context&& other) noexcept :
        m_context(std::move(other.m_context)),
        m_context_type(std::exchange(
                        other.m_context_type, -1)) {}
    resume_apartment_context& operator=(
        resume_apartment_context const&) = default;
    resume_apartment_context& operator=(
        resume_apartment_context&& other) noexcept
    {
        m_context = std::move(other.m_context);
        m_context_type =
            std::exchange(other.m_context_type, -1);
        return *this;
    }

    bool valid() const noexcept
    {
        return m_context_type >= 0;
    }

    com_ptr<IContextCallback> m_context =
        try_capture<IContextCallback>(
            WINRT_IMPL_CoGetObjectContext);
    int32_t m_context_type = get_apartment_type().first;
};
```

If COM is not active, then `CoGetObjectContext` fails, and instead of throwing an exception, we just accept the failure, and `m_context` is a null pointer.

Most of the nonsense in the `resume_apartment_context` is just making sure that the `m_context_type` resets to –1 when the `m_context` is moved out. It would be a lot simpler if we had access to the `movable_primitive` template type we put together some time ago.

We also have to teach the `resume_apartment()` function to deal with the case where COM is not running.

```
inline auto resume_apartment(
    resume_apartment_context const& context,
    coroutine_handle<> handle)
{
    WINRT_ASSERT(context.valid());
    if ((context.m_context == nullptr) ||
        (context.m_context ==
            try_capture<IContextCallback>(
                WINRT_IMPL_CoGetObjectContext)))
    {
        handle();
    }
    else if (context.m_context_type == 1 /* APTTYPE_MTA /)
    {
        resume_background(handle);
    }
    else if (is_sta_thread())
    {
        resume_apartment_on_threadpool(m_context, handle);
    }
    else
    {
        resume_apartment_sync(m_context, handle);
    }
}
```

If COM is not running, then every thread is the implicit MTA, and we can resume the coroutine anywhere.

Next time, we'll look at another hole in our framework: What happens when things go wrong?