How can I monitor changes to the reference count of a C++/WinRT object?

devblogs.microsoft.com/oldnewthing/20220225-00

February 25, 2022

Raymond Chen

Say you're debugging your C++/WinRT object and you want to keep an eye on its reference count, perhaps because you're tracking down a memory leak. How can you do that?

For concreteness, let's say we've got these objects.

```
// A local object without a projection
struct Faucet : winrt::implements<Faucet, IFaucet>
{
    bool is_dripping = false;

    Faucet()
    {
        ... construct the faucet ...
    }

    ... other methods ...
};

// A projected class
namespace winrt::Fixtures::implementation
{
    struct Lamp : LampT<Lamp>
    {
        bool is_on = false;

        Lamp()
        {
            ... construct the lamp ...
        }

        ... other methods ...
    };
}
```

If you are the caller of `make_self`, you can inspect that result to find the reference count.

```
auto faucet = winrt::make_self<Faucet>();
auto lamp = winrt::make_self<implementation::Lamp>();
```

For our purposes, `make_self` is convenient because it gives you a pointer to the implementation class, which makes it easy to extract the reference count. You can see it in the debugger:

Name	Value
◢ faucet	0x00d4bab8 {…}
◢ [winrt::impl::heap_implements<Faucet>]	{…}
◢ Faucet	{…}
◢ winrt::implements<Faucet, IFaucet>	{…}
winrt::impl::producers_base<Faucet, std::tuple<IFaucet> >	{…}
◢ winrt::impl::root_implements<Faucet, std::tuple<IFaucet> >	{m_references=0x00000001 }
winrt::impl::root_implements_composing_outer<0>	{…}
winrt::impl::root_implements_composable_inner<Faucet, 0>	{…}
winrt::impl::module_lock_updater<1>	{…}
__vfptr	0x004b4464 {…}
m_references	0x00000001 ←
IUnknown	{…}
[Raw View]	{m_ptr=0x00d4bab8 {…} }

You get a similar view for `lamp`.

From here, you can right-click the `m_references` and say *Break When Value Changes*.

If you are more of a roll-up-your-sleeves kind of person, you can extract the address of that reference count variable from the Immediate window:

&faucet.m_ptr->m_references 0x00d4babc 0x00000001

And then you can create a data breakpoint that triggers when the reference count changes.

If you're not so lucky and the object was created via projection or `make`, then what comes out is an interface pointer, not a pointer to the concrete object. So how do you get a pointer to the concrete object?

My trick is to set a breakpoint on the constructor. In the constructor, you have the `this` pointer, and you can follow the same cookbook above to get to the `m_references`.

If you're really unlucky, the constructor was optimized out. You can ask the compiler not to optimize out the constructor by marking it as `noinline`.

```
// or __attribute__((noinline)) if that's what your compiler prefers
__declspec(noinline) Faucet()
{
    ... construct the faucet ...
}
```

The last wrinkle is that you may see a write to `m_references` that comes from `make_weak_ref` instead of the usual `AddRef` and `Release`. C++/WinRT uses the same trick that WRL uses to squeeze a weak reference and a reference count into a single integer: If no weak reference has been created, then the `m_references` is the actual reference count. But once a weak reference is created, then `m_references` becomes a pointer to the weak reference, and the reference count moves into the weak reference.

When that happens, you want to double-click the call stack entry for `make_weak_ref`, expand the `weak_ref` variable, find the `m_strong` and do another *Break When Value Changes*. (The corresponding immediate expression is `&weak_ref.m_ptr->m_strong`.)

Raymond Chen

Follow