Peeking inside C++/CX weak references

devblogs.microsoft.com/oldnewthing/20200504-00

May 4, 2020

Raymond Chen

Let's hope you never need to do this, but if you are forced to debug code written in C++/CX, and you have a C++/CX weak reference, say because you pulled it out of a C++/CX delegate, and you want to know what it actually refers to, well, here goes.

```
0:003> $ 64-bit version
0:003> dps 00000203`a8773600
00000203`a8773600  00007ffd`b63d6600
wincorlib!Platform::Details::ControlBlock::`vftable'
00000203`a8773608  00000006`0000006d ⇐ reference counts
00000203`a8773610  00000203`a7edb710 ⇐ target

0:003> $ 32-bit version
0:003> dps 18773600
18773600  163d6600 wincorlib!Platform::Details::ControlBlock::`vftable'
18773604  00000006 ⇐ weak reference count
18773608  0000006d ⇐ strong reference count
1877360c  17edb710 ⇐ target
```

You can find this structure in the header file `vccorlib.h`:

```
namespace Platform { namespace Details
{
    class ControlBlock sealed : public __abi_IWeakReference
    {
    private:
        volatile long __weakRefCount;
        volatile long __strongRefCount;
        __abi_IUnknown* __target;
        ...
    }
```

In general, weak references tend to rely on a *control block* which keeps track of the number of active references. In the case of C++/CX, the control block consists of the following:

- A pointer-sized vtable for exposing the methods of *IWeakReference*.
- A 32-bit count of outstanding weak references.
- A 32-bit count of outstanding strong references.

- A pointer to the target of the weak reference, if still valid.

If the target no longer exists, then the strong reference count is zero and the target pointer is null.

In our case, we have a non-null target, so we can pull it out and find the target of the weak reference.

```
0:003> dps 00000203`a7edb710 l1
00000203`a7edb710  00007ffd`aee74450 contoso!Contoso::Widget::`vftable'
```

In this case, it's a `Contoso:: Widget`.

Raymond Chen

Follow