Getting a strong reference from the this pointer too late

devblogs.microsoft.com/oldnewthing/20230526-00

May 26, 2023

Raymond Chen

It is a standard pattern for functions that are coroutines to promote the `this` pointer to a strong reference (either a COM strong reference or a `shared_ptr`), so that the object won’t be destructed while the coroutine is suspended. But it might be too late.

Consider the following example:

```
struct MyObject : winrt::implements<MyObject, winrt::IInspectable>
{
  MyObject() = default;
  ~MyObject() = default;

  winrt::Widget::Closed_revoker m_revoker;

  void RegisterForWidgetEvents(Widget const& widget)
  {
      m_revoker = widget.Closed(winrt::auto_revoke,
        { this, &MyObject::OnWidgetClosed });
  }

  winrt::fire_and_forget OnWidgetClosed(Widget const& sender, winrt::IInspectable
const&)
  {
    auto lifetime = get_strong();

    co_await DoStuffAsync();
    co_await DoMoreStuffAsync();
  }
};
```

The idea here is that we register for the Widget’s `Closed` event with a raw pointer. When the event is raised, the handle immediately promotes the raw pointer to a strong reference, so that the `MyObject` does not destruct during the two asynchronous calls that follow.

But there’s still a race condition:

Thread 1	Thread 2

	Widget closes
Last reference released	
	`Widget::OnWidgetClosed` begins
Destruction begins	
`m_revoker` unregisters handler	
	`get_strong()`

If the last reference is released before the `Widget::OnWidgetClosed` method reaches the `get_strong()`, then the `get_strong()` method runs against an object that has already started destructing. It will nevertheless produce a strong reference and increment the reference count, but that reference count does not have the power of time travel. The destructor is already running; you incremented the reference count too late. The result is a mysterious crash.

A similar problem exists with `std::shared_ptr`:

```
struct MyObject : std::enable_shared_from_this<MyObject>
{
  MyObject() = default;
  ~MyObject() = default;

  winrt::Widget::Closed_revoker m_revoker;

  void RegisterForWidgetEvents(Widget const& widget)
  {
      m_revoker = widget.Closed(winrt::auto_revoke,
        { this, &MyObject::OnWidgetClosed });
  }

  winrt::fire_and_forget OnWidgetClosed(Widget const& sender, winrt::IInspectable
const&)
  {
    auto lifetime = shared_from_this();

    co_await DoStuffAsync();
    co_await DoMoreStuffAsync();
  }
};
```

Thread 1	Thread 2
	Widget closes

Last reference released	
	`Widget::OnWidgetClosed` begins
Destruction begins	
`m_revoker` unregisters handler	
	`shared_from_this()`

The call to `shared_from_this()` throws `std::bad_weak_ptr` because the weak pointer cannot be converted to a `shared_ptr`.

In both cases, the problem is that the `OnWidgetClosed` callback is registered with a raw pointer. Instead, use a weak pointer and try to promote it to a strong pointer in the callback.

```
// C++/WinRT
void RegisterForWidgetEvents(Widget const& widget)
{
  m_revoker = widget.Closed(winrt::auto_revoke,
    [weak = get_weak()](auto&& sender, auto&& args)
    {
      if (auto strong = weak.get()) {
        strong->OnWidgetClosed(sender, args);
      }
    });
}

// C++/WinRT alternate version
void RegisterForWidgetEvents(Widget const& widget)
{
  m_revoker = widget.Closed(winrt::auto_revoke,
    { get_weak(), &MyObject::OnWidgetClosed });
}

// C++ standard library
void RegisterForWidgetEvents(Widget const& widget)
{
  m_revoker = widget.Closed(winrt::auto_revoke,
    [weak = weak_from_this()](auto&& sender, auto&& args)
    {
      if (auto strong = weak.lock()) {
        strong->OnWidgetClosed(sender, args);
      }
    });
}
```

C++/WinRT provides a helper constructor that does the `auto strong = weak.get()` thing automatically.

Since weak pointers will not promote to strong/shared pointers once the last strong/shared reference is destructed, you don’t have the race condition where the callback tries to do something with an object that has begun destructing.

Thread 1	Thread 2
	Widget closes
Last reference released (weak pointers are now expired)	
	`Widget::OnWidgetClosed` begins
Destruction begins	
`m_revoker` unregisters handler	
	`weak.get()` fails