C++/WinRT event handlers that are lambdas with weak pointers to the parent class, part 1

devblogs.microsoft.com/oldnewthing/20230531-00

May 31, 2023

Raymond Chen

C++/WinRT provides multiple ways of auto-implementing delegates.

```
// Function pointer
e.add(&MyClass::StaticMemberFunction);

// Raw pointer + non-static member function pointer
e.add({ this, &MyClass::MemberFunction });

// Strong pointer + non-static member function pointer
e.add({ get_strong(), &MyClass::MemberFunction });

// Weak pointer + non-static member function pointer
e.add({ get_weak(), &MyClass::MemberFunction });

// Lambda
e.add(lambda);
```

Internally, all of the constructors are just shorthand for lambdas:

```
template <typename F> delegate_base(F* handler) :
    delegate_base(
        [=](auto&& ... args)
        { return handler(args...); }) {}

template <typename O, typename M> delegate_base(O* object, M method) :
    delegate_base(
        [=](auto&& ... args)
        { return ((*object).*(method))(args...); }) {}

template <typename O, typename M> delegate_base(com_ptr<O>&& object, M method) :
    delegate_base(
        [o = std::move(object), method](auto&& ... args)
        { return ((*o).*(method))(args...); }) {}

template <typename O, typename M> delegate_base(winrt::weak_ref<O>&& object, M
method) :
    delegate_base(
        [o = std::move(object), method](auto&& ... args)
        { if (auto s = o.get()) { ((*s).*(method))(args...); } }) {}
```

One pattern that is not covered by the above is the case of a parameterized callback.

```
void MyClass::OnSomething(IInspectable const& sender,
                          IInspectable const& args,
                          int otherData)
{
    ...
}
```

You want to register `OnSomething` as an event handler in multiple places, with each place having a different `otherData`.

```
// When the Widget.Something event occurs,
// call the OnSomething method with a specific otherData.
void MyClass::RegisterSomething(int otherData)
{
    // Doesn't work
    widget.Something({ get_weak(), &MyClass::OnSomething, otherData });
}
```

This doesn’t work because a pointer to member function doesn’t have a place to stick extra information. Instead, you have to use a lambda:

```
// When the Widget.Something event occurs,
// call the OnSomething method with a specific otherData.
void MyClass::RegisterSomething(int otherData)
{
    // Raw pointer version
    widget.Something(
        [this, otherData]
        (auto&& sender, auto&& args)
        {
            this->OnSomething(sender, args, otherData);
        });
}
```

This version captures `this` as a raw pointer, but we can teach it about the other two patterns:

```
void MyClass::RegisterSomething(int otherData)
{
    // Strong pointer version
    widget.Something(
        [strong = get_strong(), otherData]
        (auto&& sender, auto&& args)
        {
            strong->OnSomething(sender, args, otherData);
        });
}

void MyClass::RegisterSomething(int otherData)
{
    // Weak pointer version
    widget.Something(
        [weak = get_weak(), otherData]
        (auto&& sender, auto&& args)
        {
            if (auto strong = weak.get()) {
                strong->OnSomething(sender, args, otherData);
            }
        });
}
```

In practice, the work of the lambda is often inlined, so you end up with the following:

```
// When the Widget.Something event occurs,
// call the OnSomething method with a specific otherData.
void MyClass::RegisterSomething(int otherData)
{
    // Raw pointer version
    widget.Something(
        [this, otherData]
        (auto&& sender, auto&& args)
        {
            DoThing1(sender);
            DoThing2(args);
            DoThing3(otherData);
        });
}

void MyClass::RegisterSomething(int otherData)
{
    // Strong pointer version, with bonus "this" capture
    // for convenience.
    widget.Something(
        [strong = get_strong(), this, otherData]
        (auto&& sender, auto&& args)
        {
            DoThing1(sender);
            DoThing2(args);
            DoThing3(otherData);
        });
}

void MyClass::RegisterSomething(int otherData)
{
    // Weak pointer version
    widget.Something(
        [weak = get_weak(), this, otherData]
        (auto&& sender, auto&& args)
        {
            if (auto strong = weak.get()) {
                DoThing1(sender);
                DoThing2(args);
                DoThing3(otherData);
            }
        });
}
```

In the first two cases, you just write a lambda that does what you want to do. The third case, however, is annoying: You have to perform an extra step to resolve the weak reference. This extra step was handled by the C++/WinRT “weak pointer + non-static member function pointer” wrapper, but the lambda wrapper doesn’t provide the same courtesy.

Next time, we’ll look at extending the courtesy to lambdas.