My class derives from std::enable_shared_from_this, but shared_from_this() doesn't work

devblogs.microsoft.com/oldnewthing/20220720-00

July 20, 2022

Raymond Chen

If you make a class `T` that derives from `std::enable_shared_from_this<T>`, then the creation of a `std::shared_ptr` to that class will activate the `shared_from_this()` method to return a `shared_ptr` that shares ownership with the originally-created `std::shared_ptr`.

The catch is that the `shared_ptr` constructor and `enable_shared_from_this` are in cahoots, and the `shared_ptr` must be able to access the `enable_shared_from_this` in order to finish the job. This requires that you *publicly* derive from `std::enable_shared_from_this`:

```
class MyClass : public std::enable_shared_from_this<MyClass>
{
    ...
};
```

If you forget the `public` keyword, then the base class defaults to `private`, and the secret signal between `shared_ptr` and `enable_shared_from_this` does not get through.

Here's how `enable_shared_from_this` and `shared_ptr` work together. Note that I've ignored edge cases; the idea here is to give the basic idea so you can diagnose `enable_shared_from_this` issues yourself.

```
template<typename T>
struct enable_shared_from_this
{
    shared_ptr<T> shared_from_this()
    { return shared_ptr<T>(weak_this); }

    weak_ptr<T> weak_this;
};

template<typename T>
struct shared_ptr
{
    shared_ptr(T* p) : ptr(p)
    {
        if (T derives from enable_shared_from_this) {
            ptr->weak_ptr = *this;
        }
    }

    T* ptr;
    /* other stuff */
};
```

When a `shared_ptr` is created, it snoops at the managed object to see if it derives from `enable_shared_from_this`. If so, then it sets the `weak_ptr` to hold a weak pointer to the shared object. When you later ask for a `shared_from_this()`, it promotes this weak pointer to a shared pointer and returns it.

Okay, so we already see some consequences and pitfalls:

First of all, if you fail to derive *publicly* from `enable_shared_from_this`, the feature simply fails silently. There is no diagnostic that says, “Hey, like, you’re deriving from `enable_shared_from_this`, but you did it privately, so it’s not going to work.”[1]

Second, notice that the weak pointer is set only when the object is placed inside a `shared_ptr`, which happens *after* the shared object has been constructed. This means that you cannot use `shared_from_this()` in your constructor.

Third, if the object is not wrapped inside a `shared_ptr` at all, then `shared_from_this()` will always fail. For example, if somebody constructs the object on the stack, or via `new` or `make_unique`, it will not be controlled by a `shared_ptr`.

There are so many ways `enable_shared_from_this` can go wrong. Next time, we’ll see what we can do to guard against them.

[1] Maybe it’s possible to add a diagnostic to `shared_from_this()`. I wonder if the shared type is required to be complete by that point.

Raymond Chen

Follow