Creating a lazy-start C++/WinRT coroutine from an eager-start one, part 2

devblogs.microsoft.com/oldnewthing/20220914-00

September 14, 2022

Raymond Chen

Last time, we created a lazy-start C++/WinRT coroutine wrapper around a standard `IAsyncAction` or `IAsyncOperation`, which is an eager-start coroutine. We did it by sharing a kernel handle with the coroutine and having the coroutine wait for the handle to be signaled before beginning its work.

But we can do better, if we are willing to dig down a little bit.

The trick here is to have the coroutine suspend itself, and then manually resume it.

```
template<typename Make>
auto MakeLazy(Make make) -> decltype(make())
{
    struct suspender
    {
        void resume() { handle(); }

        bool await_ready() { return false; }
        void await_suspend(std::experimental::coroutine_handle<> h)
        { handle = h; }
        void await_resume() { }
    private:
        std::experimental::coroutine_handle<> handle;
    };

    suspender suspend;
    auto currentTask = [](auto start, auto make, auto& suspend)
        -> decltype(make()) {
        co_await suspend;
        co_return co_await make();
    }(std::move(start), std::move(make), suspend);

    // Resume the coroutine
    suspender.resume();
    return currentTask;
}
```

The idea here is to create a custom awaitable object which saves the coroutine handle, which can be resumed by an explicit call to the `resume()` method.

Recall that what makes a C++ coroutine lazy-start is that it chooses to make its `initial_suspend` method return a suspending awaiter. `IAsyncAction` and `IAsyncOperation` are eager-start, so the C++/WinRT libray's `initial_suspend` is `suspend_never`, thereby allowing the coroutine to begin executing its body. By adding an immediate suspension point as the first thing in the body, we are sort of retroactively changing that `suspend_never` into a suspension. If you look at one of the rewrite steps in the coroutine transformation, you'll see that the coroutine function body comes immediately after the `initial_suspend`, so our immediate suspension is functionally equivalent to rewriting the `initial_suspend`.

We will expand upon this idea next time, when we work on serializing asynchronous operations in C++/WinRT.

Raymond Chen

Follow