In C++/WinRT, how can I await multiple coroutines and capture the results?, part 1

devblogs.microsoft.com/oldnewthing/20240110-00

January 10, 2024

Raymond Chen

The `winrt::when_all` function in C++/WinRT is not a complicated beast. It just awaits each of its parameters. Some time ago, we looked at how it works and how you could use it to await all the coroutines in a container or range. But all of those examples just throw away the results. What if we also want to gather the results?

Here's what doesn't work:

```
auto op1 = DoSomething1Async();
auto op2 = DoSomething2Async();

co_await winrt::when_all(op1, op2);

// These don't return proper results.
auto result1 = op1.GetResults();
auto result2 = op2.GetResults();
```

The problem is that the `co_await` inside `when_all` calls `GetResults()` in order to produce the result of the `co_await`. And `GetResults()` produces valid results only the first time it is called after completion.

One way to solve this problem is to use a different awaiter whose `await_resume()` doesn't call `GetResults()`, leave it to the caller to fetch the results when they are ready.

```
template<typename Async>
struct simple_async_awaiter
{
    simple_async_awaiter(Async const& async) : async(async) {}

    Async const& async;

    bool await_ready() const noexcept { return false; }

    void await_suspend(std::coroutine_handle<> handle) const {
        async.Completed([handle](auto&&...) { handle(); });
    }

    void await_resume() const noexcept {
        /* return async.GetResults(); */
    }
};
```

We return `void` from `await_resume()`, which means that `co_await` returns `void`, and you can then fetch the results by calling `GetResults()`. (We also don't bother preserving the COM apartment context, and we don't propagate cancellation, but the point here was to show that we skipped the `GetResults()` inside `await_resume()`.)

But this is an awful lot of work, particularly because you have to replicate all of the stuff that C++/WinRT does behind the scenes in its awaiter. Maybe we can find something simpler.

We'll continue our exploration next time.