Rough edges in the when_all coroutine, part 1: Empty parameter list

devblogs.microsoft.com/oldnewthing/20200903-00

September 3, 2020

Raymond Chen

Last time, we looked at how we could write our own `when_all` coroutine that completes when all of its parameters have completed. It turned out to be very simple, thanks to C++ fold expressions.

```
template <typename... T>
Windows::Foundation::IAsyncAction when_all(T... async)
{
    (co_await async, ...);
}
```

But there's a problem in the edge case where the caller passes no parameters at all. This may not be something you do intentionally, but it could fall out of some other template metaprogramming algorithm where you discover that there is nothing to do, and the parameter pack expansion is empty. Or maybe it's the base case of some recursive algorithm.

In the case where there are no parameters, `when_all` is vacuously complete (nothing to wait for), but you get a compiler warning:

```
control reaches end of non-void function
```

What's that about?

If there are no parameters at all, then the fold expression has nothing to `co_await`, and per the language specification (**[tab:temp.fold.empty]**), a folded comma operator with no arguments, the result is `void()`. The effect is therefore this:

```
Windows::Foundation::IAsyncAction when_all()
{
    void();
}
```

And now we see the problem. Since there is no `co_await`, `co_return` or `co_yield` anywhere in the body, the function is *not a coroutine*. It's just a regular function that returns an `IAsyncAction`, and it forgot to return one!

We can fix this by adding a `co_return`. In the case where the parameter list is nonempty, the `co_return` is redundant but harmless, because falling off the end of a coroutine which produces a `void` is equivalent to performing a `co_return` just before the close-brace. The purpose of the explicit `co_return` is to ensure correct behavior when the parameter list is empty.

Next time, another edge case.

Raymond Chen

Follow