How to wait for multiple C++ coroutines to complete before propagating failure, false hope

devblogs.microsoft.com/oldnewthing/20230628-00

June 28, 2023

Raymond Chen

Last time, we tried to use a lambda in conjunction with a parameter pack fold expression in order to iterate over the parameter pack. Unfortunately, a lambda creates a nested function call, and the lambda coroutine's return type was `IAsyncAction`, which has a policy in C++/WinRT of resuming in the same apartment context, even if the underlying coroutine completed in some other context. This erases any apartment-changing effects of the original awaitable, which is a behavior change from `when_all`.

What's a parameter that represents a variable-length thing that I can still iterate over? I know: `initializer_list`!

```
template<typename... T>
IAsyncAction when_all_complete(T... asyncs)
{
    return when_all_complete_list({ asyncs... });
}

template<typename. T>
IAsyncAction when_all_complete_list(
    std::initializer_list<T> asyncs)
{
    std::exception_ptr eptr;

    for (auto&& async : asyncs) {
        try {
            co_await async;
        } catch (...) {
            if (!eptr) {
                eptr = std::current_exception();
            }
        }
    }

    if (eptr) std::rethrow_exception(eptr);
}
```

We take the parameter pack and pass it to a helper function `when_all_complete_list`, which reinterprets it as a `std::initializer_list`. The nice thing aobut a `std::initializer_list` is that you can iterate over it, and in the loop body, we put our `try`/`co_await`/`catch` boilerplate.

I thought I had done it. It passed a basic test like

```
IAsyncAction Sample(int s)
{
    co_await winrt::resume_after(std::chrono::seconds(s));
    printf("Finished after %d seconds\n", s);
}

IAsyncAction test()
{
    co_await when_all_complete(
        Sample(1), Sample(2), Sample(3)
    );
    printf("All complete\n");
}
```

But then I realized that I had blown it.

Do you see how I fooled myself?

The conversion of the parameter pack to a `std::initializer_list` succeeds only if all of the types in the parameter pack are the same. And it so happens that in my basic test, all of the types are the same, namely `IAsyncAction`. If I had tried

```
    co_await when_all_complete(
        winrt::resume_background(),
        Sample(1), Sample(2), Sample(3)
    );
```

then I would have gotten a compiler error complaining that it couldn't make a `std::initializer_list` out of a heterogeneous list.

Well that was unfortunate.

We'll try again next time.

Bonus chatter: Another problem is that constructing the initializer list requires that the `T`'s all be copyable.