In C++/WinRT, how can I await multiple coroutines and capture the results?, part 2

devblogs.microsoft.com/oldnewthing/20240111-00

January 11, 2024

Raymond Chen

Instead of replacing the awaiter so it doesn't retrieve the results, we can go ahead and collect the results, and then return them. The Windows Runtime doesn't have a convenient way to return a strongly-typed heterogeneous collection. Structures must be declared in metadata, and returning a vector of `IInspectable`s is not strongly-typed.

Fortunately, we can use our friend `simple_task`, which has since been added to the Windows Implementation Library as `wil::task`.

```
template<typename... Results>
wil::task<std::tuple<Results>>
when_all_with_results(
    winrt::Windows::Foundation::IAsyncOperation<Results>... asyncs)
{
    co_return std::make_tuple(co_await asyncs...);
}

auto [result1, result2] =
    co_await when_all_with_results(Do1Async(), Do2Async());
```

We wish we could have written

```
template<typename... Asyncs>
wil::task<auto>
when_all_with_results(Asyncs... asyncs)
{
    co_return std::make_tuple(co_await asyncs...);
}
```

but there is currently no facility in the C++ language for this sort of weirdo template placeholder usage.

The above formulation does limit you to `IAsyncOperation<T>`, so you cannot use other awaitables with `when_all_with_results`, like `IAsyncOperationWithProgress<T, P>`. Adding `IAsyncOperationWithProgress<T, P>` support isn't so bad, because the result type is available

from both `IAsyncOperation<T>` and `IAsyncOperationWithProgress<T, P>` by checking the return type of `GetResult()`.

```
template<typename... Asyncs>
wil::task<std::tuple<
    decltype(std::declval<Asyncs>().GetResults())...>>
when_all_with_results(Asyncs... asyncs)
{
    co_return std::make_tuple(co_await asyncs...);
}
```

Or, taking advantage of trailing return types so we don't need to go through the hassle of `declval`:

```
template<typename... Asyncs>
auto
when_all_with_results(Asyncs... asyncs) ->
    wil::task<std::tuple<
        decltype(asyncs.GetResults())...>>
{
    co_return std::make_tuple(co_await asyncs...);
}
```

Extending support to other types of awaitables, such as `wil::task`, means having to fire up a lot of infrastructure to figure out what the `co_await` return type is.

```
template<typename T>
using await_result = decltype(std::declval<
        awaiter_finder::type<T>>().await_resume());

template<typename... Asyncs>
wil::task<std::tuple<await_result<Asyncs>...>>
when_all_with_results(Asyncs... async)
{
    co_return std::make_tuple(co_await async...);
}
```

Great, you solved one problem but introduced at least two new ones.

First problem is that one of these awaitables might produce a C++ reference. This wasn't a problem with `IAsyncOperation`, since that never produces a C++ reference, but arbitrary awaitables might do that. Another problem is that one of the `async` values might be an awaitable that completes with `void`. You can't put a `void` inside a tuple.

We'll look more closely at these problems next time.