Creating an already-completed asynchronous activity in C++/WinRT, part 6

devblogs.microsoft.com/oldnewthing/20240716-00

July 16, 2024

Raymond Chen

Last time, we finished making generalized `MakeCompleted` and `MakeFailed` functions for creating already-completed asynchronous activities in C++/WinRT. We left off with the observation that it's annoying having to write out the template type parameter all the time:

```
winrt::Windows::Foundation::IAsyncOperationWithProgress<bool, WidgetSaveProgress>
    SaveChanges()
{
    // The widget is immutable, so there are no changes to save.
    return MakeCompleted<
        winrt::Windows::Foundation::IAsyncOperationWithProgress<
            bool, WidgetSaveProgress>>(true);
}
```

We can avoid the extra typing by returning a proxy object and let the conversion operator tell us what interface the caller wants.

```
template<typename... Result>
struct async_completed
{
    static_assert(sizeof...(Result) <= 1);
    std::tuple<std::decay_t<Result>...> m_result;

    async_completed(Result&&... result) :
        m_result(std::forward<Result>(result)...) {}

    template<typename Async>
    Async as() {
        if constexpr (sizeof...(Result) == 0) {
            co_return;
        } else {
            co_return std::move(std::get<0>(m_result));
        }
    }

    template<typename Async>
    operator Async() { return as<Async>(); }
};

template<typename Error>
struct async_failed
{
    std::decay_t<Error> m_error;

    async_failed(Error&& error) :
        m_error(std::forward<Error>(error)) {}

    template<typename Async>
    operator Async() {
        (void) co_await winrt::get_cancellation_token();
        throw std::move(m_error);
    }
};
```

Now you can let the proxy convert to the desired type.

```
winrt::Windows::Foundation::IAsyncAction
    SaveAsync()
{
    return async_completed();
}

winrt::Windows::Foundation::IAsyncOperation<bool>
    SaveAsync()
{
    return async_completed(false);
}

winrt::Windows::Foundation::IAsyncAction
    SaveAsync()
{
    return async_failed(winrt::hresult_access_denied());
}
```

Now, for cases like this, the `async_completed` and `async_failed` are awfully awkward-looking. The more natural way to write the methods would be

```
winrt::Windows::Foundation::IAsyncAction
    SaveAsync()
{
    co_return;
}

winrt::Windows::Foundation::IAsyncOperation<bool>
    SaveAsync()
{
    co_return false;
}

winrt::Windows::Foundation::IAsyncAction
    SaveAsync()
{
    (void) co_await winrt::get_cancellation_token();
    throw winrt::hresult_access_denied();
}
```

But sometimes you might want to create a failed or completed asynchronous activity outside of a coroutine, and for those cases, `async_completed` and `async_failed` might be handy.

But wait, we're firing up the coroutine infrastructure just to return a value or throw an exception. That feels like overkill. We'll look into a direct implementation next time.