How to get your C++/WinRT asynchronous operations to respond more quickly to cancellation, part 3

devblogs.microsoft.com/oldnewthing/20200923-00

September 23, 2020

Raymond Chen

Back in How to get your C++/WinRT asynchronous operations to respond more quickly to cancellation, part 2, I introduced a helper function that propagates cancellation of a coroutine into the coroutines it is itself awaiting, so that the whole thing can cancel faster.

The release of C++/WinRT version 2.0.200917.4 includes a new feature: Automatic propagation of cancellation. You can read the pull request for a deep dive.

The way it works for you, the application developer, is that you can call the `enable_propagation()` method on the cancellation token. If the coroutine is cancelled while it is busy `co_await`ing another coroutine, the cancellation of the coroutine is propagated into the awaited-for coroutine, thereby hastening the death of the outer coroutine.

For example:

```
IAsyncAction DoHttpThingAsync()
{
    auto cancellation = co_await get_cancellation_token();
    cancellation.enable_propagation();

    auto result = co_await httpClient.TryGetStringAsync(uri);

    if (result.Succeeded()) {
        DoSomethingWith(result.Value());
    }
}
```

Thanks to the `enable_propagation()`, if `DoHttpThingAsync` is cancelled while awaiting the result of `TryGetStringAsync`, the `TryGetStringAsync` operation is cancelled immediately rather than waiting for it to complete on its own, thereby avoiding a very lengthy network timeout.

The `enable_propagation()` method takes an optional `bool` parameter which specifies whether you want to enable cancellation propagation (`true` , the default value) or disable it (`false`). It also returns the previous setting, so you can restore it if you need to.

Cancellation propagation is now built into C++/WinRT, so we don't need the `Make-Cancellable` helper we saw in Part 2.

What's more, cancellation propagation supports `resume_on_signal` and `resume_after` , so you can cancel out of a wait operation. This is important for `resume_on_signal` : If you're cancelling the operation because the kernel object will never be signaled (e.g., because you realize that the thing you're waiting for will never happen), you need some way to get the coroutine to resume (in a canceled state) so it can clean up its resources. Otherwise, the coroutine will get stuck in a permanently-suspended state and end up leaked.

For compatibility with previous versions of C++/WinRT, automatic cancellation propagation is disabled by default. You must opt in by calling `enable_propagation()` .

Bonus chatter: The implementation of cancellation propagation is a little odd because it is optimized for the case that cancellation never occurs. Most of the expensive work happens during cancellation, with only a tiny bit of bookkeeping performed at each `co_await` . I paid for this extra cost by removing a lock from the `co_await` path in PR 171.

Bonus bonus chatter: Any awaiter can participate in cancellation propagation by being convertible to `winrt::enable_await_cancellation` (typically by inheriting from it) and implementing the `enable_cancellation()` method. Here is the breakdown for C++/WinRT awaitables:

Supports cancellation propagation	
Yes	**No**

`IAsyncXxx`
`resume_after`
`resume_on_signal`
	`apartment_context`
	`resume_background`
	`resume_foreground`
	`thread_pool`
	`wait_for_deferrals`

The thread-switching awaitables do not support cancellation propagation because there's nowhere to cancel back to. The original thread is long gone. The only way to proceed is to go forward and resume on the target thread.

Raymond Chen

Follow