Ordering asynchronous updates with coroutines, part 4: Bowing out, explicit version

devblogs.microsoft.com/oldnewthing/20210909-00

September 9, 2021

Raymond Chen

Last time, we looked at the “Everybody tries, but only one wins” pattern, in which everyone calculates a result, but only the last one gets to save it. While this does work, we noted that there’s an inefficiency: Every calculation runs to completion, even if it has been superseded.

We can address that problem by re-checking after every coroutine resumption whether we have already lost. If so, we just give up.

```
bool Widget::KeepGoingAfterAwait(uint32_t counter)
{
    std::lock_guard guard{ m_mutex };
    return counter = m_counter;
}

winrt::IAsyncAction Widget::RecalcAsync()
{
    auto lifetime = get_strong();

    uint32_t counter;
    winrt::hstring messageId;
    winrt::hstring lang;
    {
        std::lock_guard guard{ m_mutex };
        counter = ++m_counter;
        messageId = m_messageId;
        lang = m_lang;
    }

    auto resolved = co_await ResolveLanguageAsync(lang);
    if (!KeepGoingAfterAwait(counter)) co_return;
    auto library = co_await GetResourceLibraryAsync(resolved);
    if (!KeepGoingAfterAwait(counter)) co_return;
    auto message = library.LookupResourceAsync(messageId);
    if (!KeepGoingAfterAwait(counter)) co_return;

    std::lock_guard guard{ m_mutex };
    if (m_counter == counter) {
        m_message = message;
    }
}
```

After every `co_await`, we check whether our counter is still current. If not, then it means that while we were `co_await`ing, somebody else started a `RefreshAsync` which caused our refresh to become obsolete. Instead of proceeding with the work, only to reject it at the end, we just stop immediately.

The last `KeepGoingAfterAwait()` check is redundant because we're going to check one last time inside the lock, but I wrote it out anyway.

As we observed earlier, we can get rid of the locks if all accesses to the members are on a single thread.

```
bool Widget::KeepGoingAfterAwait(uint32_t counter)
{
    return counter = m_counter;
}

winrt::IAsyncAction Widget::RecalcAsync()
{
    auto lifetime = get_strong();

    uint32_t counter;
    winrt::hstring messageId;
    winrt::hstring lang;
    {
        std::lock_guard guard{ m_mutex };
        counter = ++m_counter;
        messageId = m_messageId;
        lang = m_lang;
    }

    auto resolved = co_await ResolveLanguageAsync(lang);
    if (!KeepGoingAfterAwait(counter)) co_return;
    auto library = co_await GetResourceLibraryAsync(resolved);
    if (!KeepGoingAfterAwait(counter)) co_return;
    auto message = library.LookupResourceAsync(messageId);
    if (!KeepGoingAfterAwait(counter)) co_return;

    m_message = message;
}
```

I removed the final check of the `m_counter`, since it is redundant with the `KeepGoing-AfterAwait()` that immediately precedes it.

Next time, we'll see how this pattern is already covered by existing functionality.

Raymond Chen

Follow