Creating a non-agile delegate in C++/WinRT, part 5: Waiting synchronously from a thread that may already be the right thread

devblogs.microsoft.com/oldnewthing/20200410-00

April 10, 2020

Raymond Chen

Last time, we looked at how we could build a delegate which, when invoked from a background thread, synchronously does work on a UI thread, but making the background thread wait until the UI work is complete. We did this by dispatching the work to the UI thread and performing a synchronous `get()` on the asynchronous activity.

Here's where we left things:

```
template<typename TLambda>
void RunSyncOnDispatcher(
    CoreDispatcher const& dispatcher,
    TLambda&& lambda)
{
  [&]() -> winrt::IAsyncAction
  {
    co_await winrt::resume_foreground(dispatcher);
    lambda();
  }().get();
}

deviceWatcher.Added(
    [=](auto&& sender, auto&& info)
    {
        RunSyncOnDispatcher(Dispatcher(), [&]()
        {
            viewModel.Append(winrt::make<DeviceItem>(info));
        });
    });
```

There's a catch here, though.

What if you accidentally call this from the UI thread?

In that case, `RunSyncOnDispatcher` will dispatch the work to the dispatcher (which is the current thread), and then block waiting for the work to run. But the work can't run because it needs the current thread, which is blocked waiting for the work to run.

To solve this problem, we need to check whether we are already on the target thread, in which case we just run the lambda immediately.

```
template<typename TLambda>
void RunSyncOnDispatcher(
    CoreDispatcher const& dispatcher,
    TLambda&& lambda)
{
  if (dispatcher.HasThreadAccess()) {
    lambda();
  } else {
    [&]() -> winrt::IAsyncAction
    {
      co_await winrt::resume_foreground(dispatcher);
      lambda();
    }().get();
  }
}
```

Remember, this entire week was dedicated to discussing a fringe corner case of Windows Runtime event handling. It may not be interesting in its own right, but it does demonstrate some techniques and gotchas that you may want to consider when writing your own custom coroutines.

Raymond Chen

Follow