Cancelling a Windows Runtime asynchronous operation, part 4: C++/CX with PPL, coroutine style with raw IAsyncAction and IAsyncOperation

devblogs.microsoft.com/oldnewthing/20200706-00

July 6, 2020

Raymond Chen

Last time, we looked at how task cancellation is projected in C++/CX with PPL and `co_await` with tasks. But PPL also supports awaiting directly on `IAsyncAction^` and `IAsyncOperation^` objects. Let's try it.

```
auto picker = ref new FileOpenPicker();
picker->FileTypeFilter.Append(L".txt");

auto pickerOp = picker->PickSingleFileAsync();
cancellation_token_source cts;
call<bool> do_cancel([pickerOp](bool) { pickerOp.Cancel(); });
timer<bool> delayed_cancel(3000U, false, &do_cancel);
delayed_cancel.start();

StorageFile^ file;
try {
    file = co_await pickerOp;
} catch (OperationCanceledException^) {
    file = nullptr;
}

if (file != nullptr) {
    DoSomething(file);
}
```

Observe that awaiting directly on an `IAsyncAction^` and `IAsyncOperation^` object throws a different exception from awaiting on a `task`.

You can see this in the `await_resume` for the `IAsyncInfo^` awaiter ind in `pplawait.h`:

```
template <typename _TaskTy, typename _HandlerTy>
struct _IAsync_awaiter {
    ...

    auto await_resume() {
        _VerifyStateForResultsCall(_Task->Status);
        return _Task->GetResults();
    }
};

void _VerifyStateForResultsCall(
    Windows::Foundation::AsyncStatus _Status)
{
    if (_Status == AsyncStatus::Canceled) {
        throw ::Platform::Exception::CreateException(E_ABORT);
    }
}
```

The PPL framework checks whether the status of the async operation is `Canceled`, and if so, it throws `E_ABORT`, which is represented in C++/CX as `OperationCanceled-Exception`.

So far, all of the cancellation exceptions have generated by the framework. That'll change soon. Next time, we'll look at C++/WinRT.

Raymond Chen

Follow