Cancelling a Windows Runtime asynchronous operation, part 3: C++/CX with PPL, coroutine style

devblogs.microsoft.com/oldnewthing/20200703-00

July 3, 2020

Raymond Chen

Last time, we looked at how task cancellation is projected in C++/CX with PPL and explicit continuations. But how about C++/CX with PPL and coroutines?

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

cancellation_token_source cts;
call<bool> do_cancel([cts](bool) { cts.cancel(); });
timer<bool> delayed_cancel(3000U, false, &do_cancel);
delayed_cancel.start();

StorageFile^ file;
try {
    file = co_await create_task(picker->PickSingleFileAsync(), cts.get_token());
} catch (task_canceled const&) {
    file = nullptr;
}

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

Notice that coroutines save us a lot of the hassle of setting up the `call` and `timer` because the objects live in the coroutine frame, which continues to exist until the coroutine completes.

Again, the task throws a `task_canceled` upon cancellation. This time, it's because of the `await_resume` for the task awaiter, which you can find in `pplawait.h`:

```
template <typename _Ty>
struct _Ppltask_awaiter {
    ...

    decltype(auto) await_resume() {
        return _Task.get();
    }
};
```

But wait, the PPL library also supports awaiting on raw `IAsyncAction^` and `IAsyncOperation^` objects. Next time, we'll look at what happens in that case.

Raymond Chen

Follow