Creating an apartment-aware PPL task from nothing

devblogs.microsoft.com/oldnewthing/20180801-00

August 1, 2018

Raymond Chen

In the Parallel Patterns Library (PPL) of the Concurrency Runtime, there are these things called `task`s. Some tasks are *apartment-aware*, which means that the default continuation context will execute the task continuation in the same COM apartment that queued the continuation. Otherwise, the task is not apartment-aware, which means that the default continuation context is arbitrary: The concurrency runtime will execute the task continuation in a thread of its choosing.

If you are working with objects that have thread affinity, you are operating on a single-threaded apartment (STA), and you need the continuation to run on that same thread so that you still have access to those objects.

The rule used by the Concurrency Runtime is that tasks which are derived from `IAsync-Action` or `IAsyncOperation<T>` are apartment-aware, and others are not.

Okay, so it's easy to create a non-apartment-aware completed task.

```
Concurrency::task<void> completed_non_apartment_aware_task()
{
 return Concurrency::task_from_result();
}
```

There is already a function in the Parallel Patterns Library for creating a completed task, and the result is a non-apartment-aware task.

The hard part is creating an apartment-aware completed task. Here's what I came up with:

```
Concurrency::task<void> completed_apartment_aware_task()
{
  Concurrency::create_task(Concurrency::create_async([]{}));
}
```

Working from the inside out: We start with a lambda that does nothing. We use `create_async` to wrap that lambda inside an `IAsyncAction`. Then we use `create_task` to wrap the `IAsyncAction` inside a `task`.

It's not pretty, but it works.

Now you can write things like

```
completed_apartment_aware_task()
.then([this]()
{
   // something
}).then([this](int result)
{
   // something
});
```

and all of the `something`s will run on the same apartment as the code that started the task chain.

This is particularly handy when you want to run a task conditionally on a UI thread. For the branch where you don't want a task, you still have to make one, and you want it to be apartment-aware, so that your UI code stays on the UI thread.

```
Concurrency::task<void> MaybeDoSomethingAsync()
{
  if (condition) {
    return Concurrency::create_task(...);();
  } else {
    return completed_apartment_aware_task();
  }
}
```

In the case where the condition is false, you still have to return a task, and you want it to be an apartment-aware task.

Bonus chatter: This little detour through `IAsyncAction` is necessary only if you are using `concurrency::task::then()` to attach continuations.

If you use `co_await` with `Concurrency::task`, then the `pplawait.h` header file controls how the continuation is scheduled, and it uses `task_continuation_context::get_current_winrt_context()` to schedule the continuation, which means that the task continues in the same apartment.

If you use `co_await` with C++/winrt, then the continuation runs in the same apartment, although there are special awaitable objects for explicitly moving between apartments.

Raymond Chen

Follow