“Task.Factory.StartNew” vs “new Task(…).Start”

Stephen Toub - MSFT

June 13th, 2010

With TPL, there are several ways to create and start a new task. One way is to use the constructor for task followed by a call to the Start method, e.g.

```
    new Task(…).Start();
```

and the other is by using the StartNew method of TaskFactory, e.g.

```
    Task.Factory.StartNew(…);
```

This begs the question… when and why would you use one approach versus the other?

In general, I always recommend using Task.Factory.StartNew *unless* the particular situation provides a compelling reason to use the constructor followed by Start. There are a few reasons I recommend this. For one, it’s generally more efficient. For example, we take a lot of care within TPL to make sure that when accessing tasks from multiple threads concurrently, the “right” thing happens. A Task is only ever executed once, and that means we need to ensure that multiple calls to a task’s Start method from multiple threads concurrently will only result in the task being scheduled once. This requires synchronization, and synchronization has a cost. If you construct a task using the task’s constructor, you then pay this synchronization cost when calling the Start method, because we need to protect against the chance that another thread is concurrently calling Start. However, if you use Task.Factory.StartNew, we know that the task will have already been scheduled by the time we hand the task reference back to your code, which means it’s no longer possible for threads to race to call Start, because every call to Start will fail. As such, for StartNew we can avoid that additional synchronization cost and take a faster path for scheduling the task.

There are, however, some cases where creating a new task and then starting it is beneficial or even required (if there weren’t, we wouldn’t have provided the Start method). One example is if you derive from Task. This is an advanced case and there’s typically little need to derive from Task, but nevertheless, if you do derive from it the only way to schedule your custom task is to call the Start method, since in .NET 4 the Task.Factory.StartNew will always return the concrete Task or Task<TResult> types. Another even more advanced use case is in dealing with certain race conditions. Consider the need for a task’s body to have access to its own reference, such as if the task wanted to schedule a continuation off of itself. You might try to accomplish that with code like:

```
    Task t = null;
    t = Task.Factory.StartNew(() =>
    {
        …
        t.ContinueWith(…);
    });
```

This code, however, is buggy. There is a chance that the ThreadPool will pick up the scheduled task and execute it before the Task reference returned from StartNew is stored into *t*. If that happens, the body of the task will see Task *t* as being null. One way to fix this is to separate the creation and scheduling of the task, e.g.

```
    Task t = null;
    t = new Task(() =>
    {
        …
        t.ContinueWith(…);
    });
    t.Start();
```

Now, we know that *t* will in fact be properly initialized by the time the task body runs, because we’re not scheduling it until after it’s been set appropriately.

In short, there are certainly cases where taking the “new Task(…).Start()” approach is warranted. But unless you find yourself in one of those cases, prefer TaskFactory.StartNew.

Stephen Toub - MSFT Partner Software Engineer, .NET
Follow