Async-Async: Consequences for parameters to parallel asynchronous calls

devblogs.microsoft.com/oldnewthing/20190501-00

May 1, 2019

Raymond Chen

Last time, we learned about a feature known as Async-Async which makes asynchronous operations even more asynchronous by pretending that they started before they actually did. The introduction of Async-Async is intended to be transparent to both the client and the server, provided they were following the rules to begin with. Of course, if you weren't following the rules, then you may notice some side effects.

From the client side, it means that you cannot mutate the parameters of an asynchronous operation until the operation completes. This was never permitted to begin with, but people sometimes got away with it because they "knew" that certain parameters were consumed before the initial method returned an asynchronous operation.

```
// Code in italics is wrong.

// Create three widgets in parallel.
var options = new WidgetOptions();

// Create a blue widget.
options.Color = Colors.Blue;
var task1 = Widget.CreateAsync(options);

// Create another blue widget.
var task2 = Widget.CreateAsync(options);

// Create a red widget.
options.Color = Colors.Red;
var task3 = Widget.CreateAsync(options);

// Wait for all the widgets to be created.
await Task.WhenAll(task1, task2, task3);

// Get the widgets.
var widget1 = task1.Result;
var widget2 = task2.Result;
var widget3 = task3.Result;
```

This code “knows” that the `Widget. CreateAsync` method looks at the `options.Color` *before* it returns with an `IAsyncOperation`. It therefore “knows” that any changes to the `options` after `Widget. CreateAsync` returns will not have any effect on the widget being created, so it goes ahead and reconfigures the `options` object so it can be used for the third widget.

This code does not work when Async-Async is enabled. The calls to `Widget. CreateAsync` will return immediately with fake `IAsyncOperation`s, while the real calls to `Widget. CreateAsync` are still in progress. As we saw earlier, the result of the real call to `Widget. CreateAsync` will be connected to the fake `IAsyncOperation` so that you get the result you want, but the timing has changed to improve performance. If the above code manages to change the `options. Color` to red before one of the first two real calls to `Widget. CreateAsync` reads the options, then one or both of the first two widgets will end up red rather than blue.

This is basically a case of violating one of the basic ground rules for programming: You cannot change a parameter while the function call is in progress. It’s just that for asynchronous operations, the “in progress” extends all the way through to the completion of the asynchronous operation.

It’s fine to kick off multiple asynchronous operations. Just make sure they don’t interfere with each other.

```
// Create three widgets in parallel.
var options = new WidgetOptions();

// Create a blue widget.
options.Color = Colors.Blue;
var task1 = Widget.CreateAsync(options);

// Create another blue widget.
var task2 = Widget.CreateAsync(options);

// Create a red widget.
options = new WidgetOptions();
options.Color = Colors.Red;
var task3 = Widget.CreateAsync(options);

// Wait for all the widgets to be created.
await Task.WhenAll(task1, task2, task3);

// Get the widgets.
var widget1 = task1.Result;
var widget2 = task2.Result;
var widget3 = task3.Result;
```

This time, we create a new `WidgetOptions` object for the final call to `Widget.Create-Async`. That way, each call to `Widget. CreateAsync` gets an `options` object that is stable for the duration of the call. It's okay to share the `options` object among multiple calls (like we did for the first two blue widgets), but don't change them while there is still an asynchronous operation that is using them.

Of course, once the operation completes, then you are welcome to do whatever you like to the `options`, since the operation isn't using them any more.

```
// Create three widgets in series.
var options = new WidgetOptions();

// Create a blue widget.
options.Color = Colors.Blue;
var widget1 = await Widget.CreateAsync(options);

// Create another blue widget.
var widget2 = await Widget.CreateAsync(options);

// Create a red widget.
options.Color = Colors.Red;
var widget3 = await Widget.CreateAsync(options);
```

In this case, we created the widgets in series. We changed the `options` after awaiting the result of the operation, so we know that the operation is finished and it is safe to modify the `options` for a new call.

Next time, we'll look at another consequence of Async-Async.

Raymond Chen

Follow