Async-Async: Consequences for exceptions

devblogs.microsoft.com/oldnewthing/20190503-00

May 3, 2019

Raymond Chen

As we've been learning, the feature known as Async-Async makes asynchronous operations even more asynchronous by pretending that they started before they actually did. The effect of Async-Async is transparent to properly-written applications, but if you have been breaking the rules, you may notice some changes to behavior. Today we'll look at exceptions.

```
// Code in italics is wrong.

Task task1 = null;
Task task2 = null;
try
{
    task1 = DoSomethingAsync(arg1);
    task2 = DoSomethingAsync(arg2);
}
catch (ArgumentException ex)
{
    // One of the arguments was invalid.
    return;
}

// Wait for the operations to complete.
await Task.WhenAll(task1, task2);
```

This code "knows" that the invalid parameter exception is raised as part of initiating the asynchronous operation, so it catches the exception only at that point.

With Async-Async, the call to `DoSomethingAsync` returns a fake `IAsyncOperation` immediately, before sending the call to the server. If the server returns an error in response to the operation, it's too late to report that error to the client as the return value of `DoSomethingAsync`. Because, y'know, time machine.

The exception is instead reported to the completion handler for the `IAsyncOperation`. In the above case, it means that the exception is reported when you `await` the task, rather than when you create the task.

```
try
{
    Task task1 = DoSomethingAsync(arg1);
    Task task2 = DoSomethingAsync(arg2);

    // Wait for the operations to complete.
    await Task.WhenAll(task1, task2);
}
catch (ArgumentException ex)
{
    // One of the arguments was invalid.
    return;
}
```

Again, this is not something that should affect a properly-written program, because you don't know when the server is going to do its parameter validation. It might do parameter validation before creating the `IAsyncOperation`, or it might defer doing the parameter validation until later for performance reasons. You need to be prepared for the exception to be generated at either point.

In practice, this is unlikely to be something people stumble across because Windows Runtime objects generally reserve exceptions for fatal errors, so you have no need to try to catch them.

Raymond Chen

Follow