Async-Async revisited: What about cancellation?

devblogs.microsoft.com/oldnewthing/20250212-00

February 12, 2025

We learned some time ago about Async-Async, a feature that reduces the chattiness of cross-thread asynchronous operations. One thing we didn't cover in the diagram is cancellation. How does cancellation work in Async-Async?

Recall that Async-Async works by creating two helper objects, one on the client side that pretends to be the server, and one on the server side that pretends to be the client. When the client initiates an asynchronous operation on the server, it receives an `IAsyncOperation` object immediately: This object is the client side helper. That client side helper asynchronously contacts the server-side helper object, which itself initiates the actual asynchronous operation. This call is held pending until the actual asynchronous operation completes, at which point the cross-thread call is completed with the operation results.

What happens when the operation is cancelled depends on when you cancel it.

One possibility is that you manage to cancel the operation even before the client side helper can submit the call to the server.

Client		Client Layer	Server Layer	Server
`DoSomethingAsync()`	→	create fake `IAsyncOperation`		
	⇠	return fake `IAsyncOperation`		
`put_Completed(callback)`	→	save in fake `IAsyncOperation`		
	⇠	return		
`Cancel()`	→			

	←	`callback.Invoke()`
`get_Status()`	→	
	⇠	return Cancelled
`GetResults()`	→	
	⇠	fail with `ERROR_CANCELLED`
`callback` returns	⇢	
	⇠	`Cancel()` returns
`Release()`	→	
	⇠	destroy fake `IAsync-Operation`

In the case that the operation is cancelled by the client before any call goes out to the server, the client-side fake `IAsyncOperation` just acts like a regular cancelled operation: It calls the callback to report that the operation was cancelled, and if anybody asks about the status or results of the operation, it says “It’s cancelled.”

Another case is that the client layer has already initiated the operation on the server side. In that case, the client layer cancels the pending request, but it doesn’t wait for an answer from the server. It just reports the cancellation to the client right away and lets the server cancel the underlying operation in due course.

Client		**Client Layer**		**Server Layer**	
`DoSomethingAsync()`	→	create fake `IAsyncOperation`			
	⇠	return fake `IAsyncOperation`	→	fake client	→
`put_Completed(callback)`	→	save in fake `IAsyncOperation`			⇠
	⇠	return		`put_Completed(private)`	→
				`Release()`	→
					⇠
… time passes …					

`Cancel()`	→	cancel pending call	→		
	←	`callback.Invoke()`		cancellation received	
`get_Status()`	→			`Cancel()`	→
	⇠	return Cancelled			←
`GetResults()`	→			`Release()`	→
	⇠	fail with `ERROR_CANCELLED`			⇠
`callback` returns	⇢		⇠	cancellation completes	
	⇠	`Cancel()` returns			
`Release()`	→				
	⇠	destroy fake `IAsyncOperation`			

In the case of a cancellation that takes place after the operation has already been submitted to the server, the client cancels the pending call, which the server layer interprets as a request to cancel the server-side operation.

Yet another case is that the server-side operation has completed before the client issues the cancellation. In that case, when the client layer cancels the pending request, it is told "What are you talking about? It's alredy done!" The client then apologizes and carries on.

Client		**Client Layer**		**Server Layer**	
`DoSomethingAsync()`	→	create fake `IAsyncOperation`			
	⇠	return fake `IAsyncOperation`	→	fake client	→
`put_Completed(callback)`	→	save in fake `IAsyncOperation`			⇠
	⇠	return		`put_Completed(private)`	→
				`Release()`	→
					⇠
… time passes …					

Operation completes					
					←
				`get_Status()`	→
					<--
`Cancel()`	→	cancel pending call	→	cancellation received	
	←	`callback.Invoke()`	←	`Cancel()`	→
`get_Status()`	→				<--
	<--	return Cancelled	<--	cancellation completes	
`GetResults()`	→			`GetResults()`	→
	<--	fail with `ERROR_CANCELLED`			<--
`callback` returns	-->			try to return status and results	
	<--	`Cancel()` returns		(nop – already cancelled)	
`Release()`	→			`private` returns	-->
	<--	destroy fake `IAsyncOperation`			

This behavior does not require any changes to the client or server. The fake operations introduced at the client and server behave just like a normal client and server. But if you are paying close attention, you may be able to observe some differences.

One potentially observable behavior change is that when the client cancels the operation quickly, the server-side operation might never have started. If you're correlating logs on the client and server side, you might see more client operations than server operations.

Another potentially observable behavior change is that the server may appear to receive cancellations "late" because the client's `Cancel` call returns immediately without waiting for the server-side cancellation to finish.

Yet another potentially observable behavior change is that if the server receives the cancellation after it has completed the operation, the server's logs will say that the operation ran successfully to completion, but the client's logs will say that it cancelled the operation and received a `Canceled` status.

Similarly, if the server receives the cancellation request and decides to complete the operation with `Completed` rather than cancelling, the client will still see `Canceled`.

Async-Async virtualizes out both sides of an asynchronous operation, improving overall throughput and reducing chattiness within the contract of the behavior of asynchronous operations, but it can introduce observable effects if you rely on behavior outside the strict bounds of the contract.