Why can’t you return an IAsyncAction from a coroutine that also does a co_await?

devblogs.microsoft.com/oldnewthing/20200416-00

April 16, 2020

Raymond Chen

Suppose you’re writing a coroutine, and the last thing you do is call another coroutine which has exactly the same signature as your function. You might hope to be able to pull off some sort of tail call elimination.

```
IAsyncAction DoSomethingAsync(int value);

IAsyncAction MySomethingAsync(int value)
{
  auto adjusted_value = adjust_value(value);
  return DoSomethingAsync(adjusted_value);
}
```

If there are no `co_await` or `co_return` statements in your function, then it is not compiled as a coroutine, and you can just propagate the `IAsyncAction` as your own return value.

But if you use `co_await` or `co_return`, then your function becomes a coroutine, and propagation doesn’t work:

```
IAsyncAction MySomethingAsync(int value)
{
  auto adjusted_value = co_await AdjustValueAsync(value);
  return DoSomethingAsync(adjusted_value); // doesn't compile
}
```

Instead, you have to `co_await` the final coroutine.

```
IAsyncAction DoSomethingTwiceAsync(value)
{
  auto adjusted_value = co_await AdjustValueAsync(value);
  co_await DoSomethingAsync(adjusted_value);
}
```

Why can’t you just propagate the final coroutine as the return value of your own coroutine?

You can look at it in terms of the mechanics of `co_await`: The caller is going to `co_await DoSomethingTwiceAsync()`, which means that they are going to obtain an awaiter for `IAsyncAction` and hook up their continuation to it. That awaiter is going to be managing the `IAsyncAction` that `DoSomethingTwiceAsync` returns, which is not the same as the `IAsyncAction` that the inner `DoSomethingAsync` returns.

Or you can look at it in terms of time travel: The transformation of `DoSomethingTwice-Async` into a coroutine causes the function to return an `IAsyncAction` at the point of the first suspension, whcih is at the `co_await AdjustValueAsync()` call. When the function performs the `co_await`, it returns an `IAsyncAction` that represents the remainder of the coroutine. The code that calls `DoSomethingAsync` hasn't run yet, and consequently its `IAsyncAction` does not yet exist. When the coroutine resumes, it eventually gets around to calling `DoSomethingAsync` and obtains an `IAsyncAction`. But it's far too late to return that as the return value of `DoSomethingTwiceAsync`; that function returned ages ago. You can't go back in time and say, "Oops, sorry, that's not the `IAsyncAction` I wanted to give you. Use this one instead."

Raymond Chen

Follow