A subtle way your await_suspend can access the coroutine frame when it shouldn’t

devblogs.microsoft.com/oldnewthing/20210507-00

May 7, 2021

Raymond Chen

As we learned in the very start of the series on coroutines, the `await_suspend` method cannot access the coroutine frame once it arranges for the coroutine to resume because that creates a race condition where the coroutine might already be resumed and possibly even run to completion before `await_suspend` finishes.

There’s a subtle way your `await_suspend` can access the coroutine frame that doesn’t immediately look like it’s accessing the coroutine frame.

That would be by throwing an exception.

The coroutine transformation puts the function body inside a giant `try/catch` block, and if any exception occurs, the coroutine regains control and hands the exception to the promise’s `unhandled_exception` method.

This includes the case where an exception is thrown from `await_suspend`.

If the exception is thrown before the `await_suspend` arranges for the coroutine to be resumed, then everything works as expected: The coroutine catches the exception, saves it in the promise, and then goes to its `final_suspend`. Note that the coroutine *has resumed*.

If the exception is thrown after the `await_suspend` arranges for the coroutine to be resumed, then you have a problem. Because now the coroutine is going to be resumed *twice*: once by the coroutine machinery that caught the exception and saved it in the promise, and again by whatever mechanism you used to arrange for the coroutine to be resumed.

That’s not good.

Furthermore, the `await_suspend` is racing against the resumption, and if the resumption occurs first, then the coroutine might run all the way to completion and be destroyed, and now you’re trying to save the exception into an already-destructed promise.

That’s also not good.

So don’t do that.

Raymond Chen

Follow