C++ coroutines: Tradeoffs of making the promise be the shared state

devblogs.microsoft.com/oldnewthing/20210413-00

April 13, 2021

Raymond Chen

Last time, we traced the lifetimes of the objects involved in the coroutine function. Now we can look at some of the tradeoffs we've made in our design and see how the decisions are intertwined.

Inlining the shared state into the promise itself has the advantage of avoiding an extra allocation, which is significant when coroutines are being created frequently. However, it also means that as long as there is an active task that refers to the coroutine, the entire coroutine state will remain allocated, even though most of it is not being used.

We therefore want to discourage clients from holding onto the task after the `co_await`. If the object could be `co_await`ed multiple times, then the task will need to be shared, and it will probably have a long life. For example, an initialization task would probably be stored with a long lifetime, so that every method on the object could `co_await` the initialization task to ensure that initialization was complete before proceeding.

Making the object `co_await`able only once would discourage clients from retaining the task after the `co_await`, since the object is useless after that point. Making the object `co_await`able only once also allows the result to be a move-only type, since there is only one consumer, and we can just move the result to that consumer. If the object could be `co_await`ed more than once, then each consumer would have to receive a copy.

Conversely, if we want the object to be `co_await`able multiple times, then we should keep the return value in a separate object and jettison the coroutine state as soon as possible in order to destruct the coroutine parameters as well as reclaim the memory formerly occupied by local variables and temporaries.

I find it interesting that these decisions are intertwined. Separate return object = multiply-awaitable = long lifetime. Embedded return object = singly-awaitable = short lifetime.

After writing up this discussion, it occurred to me that my simple promise implementation (which follows the embedded return object, singly-awaitable, short lifetime model) was working too hard. I'll simplify it next time.

Raymond Chen

Follow