C++ coroutines: How do I create a coroutine that terminates on an unhandled exception?

devblogs.microsoft.com/oldnewthing/20210427-00

April 27, 2021

Raymond Chen

Last time, we saw that declaring a coroutine as `noexcept` doesn't do what you think. The `noexcept` specific says that production of the coroutine does not throw an exception, but it says nothing about what happens during *execution* of the coroutine. If an exception occurs inside the coroutine, the promise's `unhandled_exception` method decides what happens.

So what can you do if you really want your coroutine to terminate on unhandled exception?

One way is to reimplement `noexcept` manually by catching all exceptions and terminating.

```
simple_task<int> GetValueAsync()
{
    try {
        co_return LoadValue();
    } catch (...) {
        std::terminate();
    }
}
```

If an exception occurs in `LoadValue()`, it is caught by the `catch (...)` and terminates the program.

You can avoid a level of indentation by moving the `try` to function scope:

```
simple_task<int> GetValueAsync() try
{
    co_return LoadValue();
} catch (...) {
    std::terminate();
}
```

This has the desired effect of terminating on unhandled exceptions, but it's kind of awkward having to wrap the function like this, and it also gets awkward if you want to turn the behavior on for only certain sections of the code.

The behavior of a coroutine in the case of an unhandled exception is left to the discretion of the coroutine promise. Some promises (like `winrt::fire_and_forget`) terminate on unhandled exceptions. Others (like our `simple_task`) stow the exception and rethrow when the task is `co_await`ed. Perhaps there's a way to configure the coroutine promise at runtime to alter its behavior. We'll look at that next time.

Raymond Chen

Follow