C++ coroutines: What happens if an exception occurs in my return_value?

devblogs.microsoft.com/oldnewthing/20210401-00

April 1, 2021

Raymond Chen

When I introduced a basic implementation of a promise type, I noted that the `return_value` method or `return_void` method is called when the coroutine performs a `co_return`. But what happens if the `return_value` or `return_void` method raises an exception?

```
void return_value(T const& v) const
{
    holder.set_result(v); // what if the copy constructor throws?
}

void unhandled_exception() const noexcept
{
    holder.set_exception(std::current_exception());
}
```

What if we take an exception trying to set the result, say because the copy constructor threw an exception? Do we have to catch the exception and convert it to a `holder.set_exception`?

```
void return_value(T const& v) const
{
    // Do I have to wrap the set_result?
    try {
        holder.set_result(v);
    } catch (...) {
        holder.set_exception(std::current_exception());
    }
}
```

Let's go back and look at the transformation that the compiler performs when it generates a coroutine:

```
return_type MyCoroutine(args...)
{
    create coroutine state
    copy parameters to coroutine frame
    promise_type p;
    auto return_object = p.get_return_object();

    try {
        co_await p.initial_suspend(); // ¹
        coroutine function body
    } catch (...) {
        p.unhandled_exception();
    }
    co_await p.final_suspend();
    destruct promise p
    destruct parameters in coroutine frame
    destroy coroutine state
}
```

The `return_value` and `return_void` happen as part of the transformation of the `co_return` statement, and that is part of the section marked *coroutine function body*. Therefore, if an exception occurs during `return_value` or `return_void`, it is caught by the `catch (...)` and is delivered to `unhandled_exception`.

In other words, the compiler already wrapped your `return_value` and `return_void` functions inside a `try` / `catch` so you don't have to.

Note however that there is no `try` / `catch` wrapped around the call to `unhandled_exception`, so that method should be careful not throw any exceptions.

Okay, so that was a brief digression on exceptions that occur when returning a value to the promise. Next time, we'll look at another improvement to our coroutine promise implementation.

Raymond Chen

Follow