winrt::fire_and_forget was too forgetful

devblogs.microsoft.com/oldnewthing/20190320-00

March 20, 2019

Raymond Chen

C++/WinRT provides a handy helper class called `winrt:: fire_ and_ forget`. It lets you specify that nobody is going to observe the result of the coroutine. This is handy because it lets you tell the compiler that the lack of observation is intentional, so it won't generate a warning.

```
winrt::Windows::Foundation::IAsyncAction DoSomething()
{
 co_await blah();
 co_await blah();
 co_await blah();
}

void OnClick()
{
 // Compiler warning on next line
 DoSomething();
}
```

The `OnClick` function calls `DoSomething`, which does a bunch of stuff asynchronously, but the `OnClick` method does not `co_await` the result, and the compiler generates a warning.

But if you say that the `DoSomething` function is fire-and-forget, then the compiler knows that there's nothing worth `co_await` ing, and not only will it not generate a warning, it also will remove a lot of the internal bookkeeping that normally occurs when you have a coroutine from which somebody might await the results.

```
winrt::fire_and_forget DoSomething()
{
 co_await blah();
 co_await blah();
 co_await blah();
}

void OnClick()
{
 // no compiler warning
 DoSomething();
}
```

There’s a catch, however, in early versions of C++/WinRT: When you told it to fire and forget, it really *forgets*. If an unhandled exception occurs, the `fire_and_forget` just swallowed the exception and didn’t tell anybody. “You told me not to bother you, so I’m not bothering you.”

You typically want to know about unhandled exceptions because they indicate a bug. We want coroutines to treat unhandled exceptions the same way that regular C++ code does: By calling `std::terminate`.

```
struct complete_asynchronously {};

namespace std::experimental
{
  template <typename ... Args>
  struct coroutine_traits<complete_asynchronously, Args ...>
  {
    struct promise_type
    {
        complete_asynchronously get_return_object() const noexcept
        {
          return{};
        }

        void return_void() const noexcept
        {
        }

        suspend_never initial_suspend() const noexcept
        {
          return{};
        }

        suspend_never final_suspend() const noexcept
        {
          return{};
        }

        void unhandled_exception() noexcept
        {
          std::terminate();
        }
      };
  };
}
```

Now we can declare that a coroutine will not be awaited, but we want unhandled exceptions to terminate the process.

```
complete_asynchronously DoSomething()
{
 co_await blah();
 co_await blah();
 co_await blah();
}
```

I personally recommend this design instead of simply dropping unhandled exceptions on the floor, because you want coroutines to treat unhandled exceptions the same way as non-coroutine code. Expressing the concept of *catch all exceptions and ignore them* is done with the conventional notation of `catch (...) { }`.

The maintainers of C++/WinRT agreed with me that the existing behavior of `winrt::fire_and_forget` was a bit too forgetful, and they made the change I recommended above. According to the pull request, the change is available starting in version 1901118.3. If you don't have that version yet, you can use `complete_asynchronously` as a stopgap.

But wait, we're not done yet. I'll continue the discussion next time.

Raymond Chen

Follow