C++ coroutines: The co_await operator and the function search algorithm

devblogs.microsoft.com/oldnewthing/20191219-00

December 19, 2019

Raymond Chen

So you're following along Kenny Kerr's blog and you get to the part where he uses `co_await` on a time duration:

```
    co_await 5s;
```

so you try it:

```
#include <chrono>
using namespace std::chrono;

winrt::IAsyncAction Delay10Seconds()
{
   co_await 10s;
   co_return;
}
```

and you get the error message

> no callable 'await_resume' function found for type 'Expression' where Expression=std::chrono::seconds

We learned that this error message means that we ended up awaiting something that can't be awaited. We were hoping that the `operator co_await` would convert the `10s` into an awaiter, but it didn't work. As a result, we ended up using the `std:: chrono:: seconds` as its own awaiter, but since it doesn't meet the requirements for an awaiter, you get an error.

As we learned last time, when you `co_await` an expression, one of the steps in obtaining an awaiter is looking for a corresponding overloaded `operator co_await` that accepts the expression. This search follows the usual mechanism for overloaded operators:

- A search is conducted for an overloaded operator declared as a member of the class.
- A search is conducted for an overloaded operator declared as a free function.

Now, the `std::chrono::seconds` doesn't implement `operator co_await` on its own, so we must search for the overloaded operator as a free function.

Tht search for a free function includes the `std::chrono` namespace, thanks to argument-dependent lookup. And it includes the namespace that is currently active, plus its parent namespaces. And it includes any names that have been imported into those namespaces.

In the case of a duration, the relevant `operator co_await` is in none of those places. It's in the `winrt` namespace.

In order for it to be found, you need to be inside a namespace (or sub-namespace) of `winrt`, or you must have imported `winrt::operator co_await` into your namespace with a `using namespace ::winrt;` statement.

If you operate entirely within C++/WinRT, then doing a `using namespace ::winrt;` is probably not a big deal. But if your code straddles the C++/WinRT and ABI worlds (or worse, straddles the C++/WinRT and C++/CX worlds, or heaven forfend, operates in all three worlds), then blanket-importing the `winrt` namespace is probably not a good idea.

Fortunately, there's a workaround.[1]

You can use `co_await winrt::resume_after(duration)` as a drop-in substitute for `co_await duration;`. This is literally what happens anyway, because the `operator co_await` definition is

```
namespace winrt
{
    inline auto operator co_await(Windows::Foundation::TimeSpan duration)
    {
        return resume_after(duration);
    }
}
```

One lesson learned from this exercise is that it may not a great idea to define a `co_await` operator outside the namespace of the object being awaited[2] because argument-dependent lookup won't find the operator if somebody tries to await the object from outside its home namespace.

Another lesson learned is that if you do define a `co_await` operator outside the namespace of the object being awaited, you should define a named function that does the work, and make your `co_await` operator call the named function. That way, people who are not `using` your namespace can still access the underlying functionality by using the named function.

[1] Another workaround is to explicitly invoke the `operator co_await` from the `winrt` namespace.

```
co_await winrt::operator co_await(duration);
```

Let us not speak of this workaround again.

[2] Corollary: It may not be a great idea to define a `co_await` operator for a language standard type.

Raymond Chen

Follow