Working around the requirement that Concurrency Runtime task results must be default-constructible and copyable

devblogs.microsoft.com/oldnewthing/20200813-00

August 13, 2020

Raymond Chen

Last time, we shed light on the hidden constraints on the result type in Concurrency Runtime tasks: The result type must have a public default constructor, and it must be copyable.

But what if your desired result type doesn't satisfy these requirements?

To work around the need for a public default constructor, you can wrap your result type inside something that does have a public default constructor, such as `std::optional`.

```
Concurrency::task<std::optional<T>>
    t([]() { return T::make(); });
```

If you produce the result from a lambda, you can just return a `T`, and a `std::optional<T>` will be constructed from it. If you produce the result from a `task_ completion_ event`, you'll have to use a `task_ completion_ event<std::optional< T>>`. The result of the task will be an `optional<T>`, and you can use the dereference operator `*` to extract the value. (This assumes that the task always completes with a value, which I assume it does, because that's what it did before you started down this path.)

To work around the need for copyability, you can wrap the result in a `std::shared_ptr<T>`. That way, there is still only one `T` object, and all the continuations get the shared copy.

And since `std::shared_ptr` has a public default constructor, if your result type falls into both categories (lacks a public default constructor, is not copyable), you can wrap it in a `std::shared_ptr<T>` and solve both problems.

Raymond Chen

Follow