A map through the three major coroutine series

devblogs.microsoft.com/oldnewthing/20210504-01

May 4, 2021

Raymond Chen

Our long national nightmare is not yet over: The three main coroutine series are now done, although that doesn't mean I'm done with coroutines.

Here's a map through the main series, at least. There is a direct route and a number of scenic routes.

Part the First: Awaitable Objects

Start the first part

C++ coroutines: Getting started with awaitable objects

- C++ coroutines: Constructible awaitable or function returning awaitable?
- C++ coroutines: Framework interop
- C++ coroutines: Awaiting an IAsyncAction without preserving thread context

C++ coroutines: Short-circuiting suspension, part 1

C++ coroutines: Short-circuiting suspension, part 2

C++ coroutines: no callable ‘await_resume’ function found for type

C++ coroutines: Defining the co_await operator

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

C++ coroutines: The problem of the synchronous apartment-changing callback

C++ coroutines: The problem of the DispatcherQueue task that runs too soon, part 1

C++ coroutines: The problem of the DispatcherQueue task that runs too soon, part 2

C++ coroutines: The problem of the DispatcherQueue task that runs too soon, part 3

C++ coroutines: The problem of the DispatcherQueue task that runs too soon, part 4

You made it to the end of the first part

Part the Second: Awaitable Signals

The early portions are optional, but things get interesting toward the end of the second part, where we build a “result holder”.

Start the second part

Creating a `co_await` awaitable signal that can be awaited multiple times, part 1

Creating a `co_await` awaitable signal that can be awaited multiple times, part 2

Creating a `co_await` awaitable signal that can be awaited multiple times, part 3

Creating a `co_await` awaitable signal that can be awaited multiple times, part 4

Creating a `co_await` awaitable signal that can be awaited multiple times, part 5

Creating a `co_await` awaitable signal that can be awaited multiple times, part 6

Creating other types of synchronization objects that can be used with `co_await`, part 1: The one-shot event

Creating other types of synchronization objects that can be used with `co_await`, part 2: The basic library

Creating other types of synchronization objects that can be used with `co_await`, part 3: Parallel resumption

Creating other types of synchronization objects that can be used with `co_await`, part 4: The manual-reset event

Creating other types of synchronization objects that can be used with `co_await`, part 5: The auto-reset event

Creating other types of synchronization objects that can be used with `co_await`, part 6: The semaphore

Creating other types of synchronization objects that can be used with `co_await`, part 7: The mutex and recursive mutex

Creating other types of synchronization objects that can be used with `co_await`, part 8: The shared mutex

Creating other types of synchronization objects that can be used with `co_await`, part 9: The shared mutex (continued)

Creating other types of synchronization objects that can be used with `co_await`, part 10: Wait for an event to clear

Creating a task completion source for a C++ coroutine: Producing a result

Creating a task completion source for a C++ coroutine: Producing a result with references

Creating a task completion source for a C++ coroutine: Producing nothing

Creating a task completion source for a C++ coroutine: Failing to produce a result

You made it to the end of the second part

Part the Third: Coroutine Promises

Start the third part

C++ coroutines: The mental model for coroutine promises

C++ coroutines: Basic implementation of a promise type

C++ coroutines: The initial and final suspend, and improving our return_value method

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

C++ coroutines: Making the promise itself be the shared state, the inspiration

C++ coroutines: Making the promise itself be the shared state, the outline

C++ coroutines: Building a result holder for movable types

C++ coroutines: Accepting types via `return_void` and `return_value`

C++ coroutines: Awaiting the `simple_task`

C++ coroutines: Managing the reference count of the coroutine state

C++ coroutines: The lifetime of objects involved in the coroutine function

C++ coroutines: Tradeoffs of making the promise be the shared state

C++ coroutines: Making it impossible to `co_await` a task twice

C++ coroutines: Getting rid of our mutex

C++ coroutines: Getting rid of our reference count

C++ coroutines: Allowing the awaiter to be destroyed while suspended

C++ coroutines: Getting rid of our atomic variant discriminator

C++ coroutines: Cold-start coroutines

C++ coroutines: Improving cold-start coroutines which complete synchronously

C++ coroutines: Associating multiple tasks with the same promise

C++ coroutines: What does it mean when I declare my coroutine as noexcept?

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

C++ coroutines: Snooping in on the coroutine body

C++ coroutines: Adding custom resume context support to our awaiter

C++ coroutines: Waiting synchronously for our coroutine to complete

C++ coroutines: Converting among tasks that use the same promise

C++ coroutines: Promise constructors

You are here

I'm not done with coroutines, but this is a road map through the three main areas.

Raymond Chen

Follow