What's the point of std::monostate? You can't do anything with it!

devblogs.microsoft.com/oldnewthing/20240708-00

July 8, 2024

Raymond Chen

C++17 introduced `std::monostate`, and I used it as a placeholder to represent the results of a coroutine that produces nothing. In the comments, Neil Rashbrook asked what you are expected to do with a `std::monostate`, seeing as has no members and only trivial member functions.

The answer is "nothing".

The purpose of `std::monostate` is to be a dummy type that does nothing. All instances are considered equal to each other. It is basically this:

```
struct monostate {};
// plus relational operators and a hash specialization
```

You can see it in libcxx (LLVM/clang), libstdc++ (gcc), and stl (msvc).

But what's the point of a class that does nothing, and which you can do nothing with?

You don't use `monostate` because you want to do something. You use `monostate` when you *don't want to do anything*.

Its original purpose was to be used as the initial type in a `std::variant` to allow it to be default-constructed in an empty state.

```
struct Widget
{
    // no default constructor
    Widget(std::string const& id);

    ⟦ other members ⟧
};

struct PortListener
{
    // default constructor has unwanted side effects
    PortListener(int port = 80);

    ⟦ other members ⟧
};

std::variant<Widget, PortListener> thingie; // can't do this
```

The `std::variant`'s default constructor default-constructs its first alternative. Since `Widget` doesn't have a default constructor, you can't put it first. And `PortListener`'s default constructor has unwanted side effects, so we don't want to put it first.

Enter `std::monostate`. You can put that guy first.

```
std::variant<std::monostate, Widget, PortListener> thingie;
```

The `thingie` default-constructs into a `monostate`. What can you do with a `monostate`? Nothing! Its job is just to be a dummy type that you can use when you are forced to provide a default-constructible type but don't want to.

In the case of a `std::variant`, you can think of inserting `std::monostate` as a way to add an "empty" state to a `variant`,[1] saving you the trouble of having to create a `std::optional<std::variant<...>>`. You can treat the `std::monostate` as the "empty" state.

In our usage of `std::monostate`, we used it as just a dummy type that stands in for `void`.[2]

[1] More precisely, a way to add *another* "empty" state to a `variant`. There is already an "empty" state for a `std::variant`, known as `valueless_by_exception`. This state is wacky, though, so you want to avoid it as much as possible. Another topic for another day.

[2] Bonus reading: Regular Void, which has stalled.