Mundane std::tuple tricks: Creating more interesting index sequences

devblogs.microsoft.com/oldnewthing/20200626-00

June 26, 2020

Raymond Chen

Last time, we wrote `offset_sequence_t`, which takes an existing sequence and adds a fixed value to each element. This is the sort of index manipulation you may want to generalize. So let's generalize it.

```
template<std::size_t F(std::size_t), typename Seq>
    struct modify_sequence;

template<std::size_t F(std::size_t), std::size_t... Ints>
struct modify_sequence<F, std::index_sequence<Ints...>>
{
 using type = std::index_sequence<F(Ints)...>;
};

template<std::size_t F(std::size_t), typename Seq>
using modify_sequence_t = typename modify_sequence<F, Seq>::type;
```

Instead of hard-coding the `+ 1` operation, we let you specify the modification operation `F`. Since this is template metaprogramming, the modification operation must be something that can be done at compile time. This means that it needs to be a `constexpr` function.

```
constexpr std::size_t add_one(std::size_t v) { return v + 1; }

// example = std::index_sequence<4, 2, 5>
using example = modify_sequence_t<
    add_one,
    std::index_sequence<3, 1, 4>>;
```

Of course, this gets annoying having to write `add_whatever` each time you want to add something, so we can generalize the `add_one` function.

```
template<int N>
constexpr std::size_t add_N(std::size_t v) { return v + N; }

// example = std::index_sequence<4, 2, 5>
using example = modify_sequence_t<
    add_N<1>,
    std::index_sequence<3, 1, 4>>;

// example2 = std::index_sequence<2, 0, 3>
using example2 = modify_sequence_t<
    add_N<-1>,
    std::index_sequence<3, 1, 4>>;
```

It's possible to write other helpers like, say, `parallel_add_t` which adds two sequences elementwise, but that sort of thing isn't needed in practice much.

Next time, we'll try to invert `tuple_element_t`.

Raymond Chen

Follow