How to clone a Windows Runtime vector in the face of possible concurrent modification, part 2

devblogs.microsoft.com/oldnewthing/20230713-00

July 13, 2023

Raymond Chen

Last time, we developed a pattern for cloning a Windows Runtime vector in the face of possible concurrent modification. Let's try to make a function out of this pattern.

```
template<typename V>
auto clone_as_vector(V const& v)
-> std::vector<decltype(v.GetAt(0))>
{
    using T = decltype(v.GetAt(0));
    std::vector<T> temp;
    uint32_t expected;
    uint32_t actual;
    do {
        expected = v.Size();
        temp.resize(expected + 1, winrt_empty_value<T>());
        actual = v.GetMany(0, temp);
    } while (actual > expected);
    temp.erase(temp.begin() + actual, temp.end());
    return temp;
}

template<typename V>
auto CloneVector(V const& v)
-> winrt::Windows::Foundation::
        Collections::IVector<decltype(v.GetAt(0))>
{
    return winrt::multi_threaded_vector(
        clone_as_vector(v));
}
```

There are some sneaky bits here.

First of all, we make `clone_as_vector` generic in that all it requires is that the `v` parameter support `GetAt()` and `GetMany()`. This allows the `clone_as_vector` to work with both `IVector` and `IVectorView`.

Second, we use an explicit trailing return type instead of just letting the compiler figure it out from the `return` statement. This allow SFINAE to remove the function from consideration, allowing us to define other versions of `clone_as_vector` that work for other collection types.[1]

This seems to work, until we run into `IVector<bool>`, because that cause the `clone_as_vector` function to create a `std::vector<bool>`.

Uh-oh.

We'll try to fix that next time.

[1] Though our trailing return type validates only that `GetAt()` works. This means that it will accidentally activate for `BindableVector` and `BindableVectorView`, which have `GetAt()` but not `GetMany()`. I guess we could fix that by adding more junk:

```
template<typename V>
auto clone_as_vector(V const& v)
-> std::enable_if_t<
    std::is_integral_v<decltype(v.GetMany(0, {}))>,
    std::vector<decltype(v.GetAt(0))>>
```