Creating a prepopulated Windows Runtime Vector from C++/WinRT without going through an explicit std::vector

devblogs.microsoft.com/oldnewthing/20240523-00

May 23, 2024

Raymond Chen

Last time, we saw that if you have to create a Windows Runtime Vector from C++/WinRT, do it as late as possible. And the extreme case of doing it as late as possible is creating it in the conversion function itself!

We'll start with the naïve slow way:

```
namespace winrt
{
    using namespace winrt::Windows::Foundation;
}

winrt::IVector<winrt::Uri> GetUris()
{
    auto v = winrt::multi_threaded_vector<winrt::Uri>();
    v.Append(winrt::Uri(L"https://microsoft.com/"));
    v.Append(winrt::Uri(L"https://contoso.com/"));
    v.Append(winrt::Uri(L"https://fabrikam.com/"));
    return v;
}
```

This follows the inefficient pattern we discussed yesterday, where we create an empty `IVector` and then use the `IVector` methods to append elements to it.

We saw last time that we could do better by working with `std::vector` first, and converting it to an `IVector` as a final step:

```
winrt::IVector<winrt::Uri> GetOptions()
{
    std::vector<winrt::Uri> v;
    v.push_back(winrt::Uri(L"https://microsoft.com/"));
    v.push_back(winrt::Uri(L"https://contoso.com/"));
    v.push_back(winrt::Uri(L"https://fabrikam.com/"));
    return winrt::multi_threaded_vector(std::move(v));
}
```

And now that you have a `std::vector`, you can take advantage of vector-specific features like `emplace_back`, which treats its arguments as parameters which will be used to construct a `winrt::Uri` object in place in the vector.

```
winrt::IVector<winrt::Uri> GetUris()
{
    std::vector<winrt::Uri> v;
    v.emplace_back(L"https://microsoft.com/");
    v.emplace_back(L"https://contoso.com/");
    v.emplace_back(L"https://fabrikam.com/");
    return winrt::multi_threaded_vector(std::move(v));
}
```

But there's a way to do this without ever having to create a `std::vector` variable at all: You can create the vector on the fly as a parameter!

```
winrt::IVector<winrt::hstring> GetUris()
{
    // Alternative 1
    return winrt::multi_threaded_vector<winrt::Uri>({
        winrt::Uri(L"https://microsoft.com/"),
        winrt::Uri(L"https://contoso.com/"),
        winrt::Uri(L"https://fabrikam.com/"),
    });

    // Alternative 2
    return winrt::multi_threaded_vector(std::vector{
        winrt::Uri(L"https://microsoft.com/"),
        winrt::Uri(L"https://contoso.com/"),
        winrt::Uri(L"https://fabrikam.com/"),
    });
}
```

You can't say

```
    return winrt::multi_threaded_vector({
        winrt::Uri(L"https://microsoft.com/"),
        winrt::Uri(L"https://contoso.com/"),
        winrt::Uri(L"https://fabrikam.com/"),
    });
```

because `multi_threaded_vector` cannot deduce the element type from the initializer list. (Maybe you can teach it and submit a pull request?)

But what you can do is take advantage of the fact that if you do specify the element type, then `multi_threaded_vector` will know that it needs a `std::vector<winrt::Uri>`, and then compiler will see from the vector constructors that the braced list is an `initializer_` `list<winrt::Uri>`, and then the compiler will see that it knows how to construct a `winrt::Uri` from a string literal, so it will automatically use the constructor to populate the initializer list. All this means that you don't have to say `winrt::Uri` around each string:

```
winrt::IVector<winrt::hstring> GetUris()
{
    return winrt::multi_threaded_vector<winrt::Uri>({
        L"https://microsoft.com/",
        L"https://contoso.com/",
        L"https://fabrikam.com/",
    });
}
```

This is still a little frustrating because it does construct a bunch of temporary `winrt::Uri`s to go into the initializer list, which are then copied into the vector. Instead, you can use the two-iterator version and let the vector do the constructing in-place.

```
winrt::IVector<winrt::hstring> GetUris()
{
    static constexpr PCWSTR uris[] = {
        L"https://microsoft.com/",
        L"https://contoso.com/",
        L"https://fabrikam.com/",
    };
    return winrt::multi_threaded_vector<winrt::Uri>(
        { std::begin(uris), std::end(uris) });
}
```