On passing iterables of iterables in the Windows Runtime

devblogs.microsoft.com/oldnewthing/20220530-00

May 30, 2022

Raymond Chen

The Windows Runtime API design guidelines recommend that methods accept iterables rather than specific collections where possible. This allows clients to pass any iterable collection of their choosing, or possibly no collection at all (like a LINQ query).

The generalization rules are:

- Arrays of T are accepted as `IIterable<T>`.
- Maps from K to V are accepted as `IIterable<IKeyValuePair<K, V>>`.

This works out well in practice because iterable collections support `IIterable`, so you can just pass the collection, and the language projection will pass the `IIterable` to the method.

Things get a little weirder if you want to pass a collection of collections. For example, suppose you have a method which wants to accept an array of maps.

```
void Something(IVector<IMap<String, String>> attributesArray);
```

Applying the above guidance means that your method looks like this:

```
void Something(IIterable<IIterable<IKeyValuePair<String, String>>> attributesArray);
```

C# supports covariance, but C++ does not. This means that in C#, you can say

```
var v = new[] {
        new Dictionary<string, string> { ["foo"] = fooValue },
        new Dictionary<string, string> { ["bar"] = barValue },
};
var result = Something(v);
```

and the language lets you pass an `IEnumerable<Dictionary<string, string>>` instead of an `IEnumerable<IEnumerable<IKeyValuePair<string, string>>>`.

C++ does not support covariance. If you try the equivalent C++, it fails.

```
// C++/CX
    using AttributeMap = Map<String^, String^>;
    AttributeMap^ v[2] = {
       ref new AttributeMap{ { "foo", fooValue } },
       ref new AttributeMap( { "bar", barValue } },
    };
    auto result = Something(v); // does not compile

// C++/WinRT
    using AttributeMap = IMap<hstring, hstring>;
    AttributeMap v[2] = {
        single_threaded_map(std::map<hstring, hstring>{ { L"foo", fooValue } },
        single_threaded_map(std::map<hstring, hstring>{ { L"bar", barValue } },
        };
    auto result = Something(v); // does not compile
```

You have to misdeclare the inner maps as iterables.

```
// C++/CX
    using AttributeMap = Map<String^, String^>;
    using IAttributeIterable = Iiterable<IKeyValuePair<String^, String^>>;
    auto d1 = ref new AttributeMap{ { "foo", fooValue } };
    auto d2 = ref new AttributeMap( { "bar", barValue } };
    auto v = ref new Vector<IAttributeIterable>({ d1, d2 });
    auto result = Something(v);

// C++/WinRT
    using IAttributeIterable = IIterable<IKeyValuePair<hstring, hstring>>;
    auto v = std::vector<IAttributeIterable>({
        std::map<hstring, hstring>({ { L"foo", fooValue } }),
        std::map<hstring, hstring>({ { L"bar", barValue } }),
    });
    auto result = Something(v);
```

Fortunately, C++/WinRT gives you a bit of help. The `winrt::params` namespace lets you write

```
    auto d1 = std::map<hstring, hstring>({ { L"foo", fooValue } });
    auto d2 = std::map<hstring, hstring>({ { L"bar", barValue } });
    auto result = Something({ d1, d2 });
```

And the appropriate conversions will happen. So C++/WinRT is roughly on par with C#. C++/CX is kind of awkward though. Sorry, C++/CX.

Bonus chatter: After writing this up, I discovered that I had already written about it, in a different way. So now you have two ways of reading about the same topic.

Raymond Chen

Follow