How to clone a Windows Runtime map in the face of possible concurrent modification, part 3

devblogs.microsoft.com/oldnewthing/20230721-00

July 21, 2023

Raymond Chen

Last time, we developed some functions for cloning a Windows Runtime map via C++/WinRT. This time, we'll do it in C++/CX, for those still using that language.

```
template<typename M>
auto clone_as_kvp_vector(M const& m)
{
    using KVP = decltype(m->First()->Current);
    std::vector<KVP> pairs;
    unsigned expected;
    unsigned actual;
    do {
        expected = m->Size;
        pairs.resize(expected + 1);
        try {
            actual = m->First()->GetMany(Platform::
                ArrayReference<KVP>(pairs.data(), expected + 1));
        } catch (Platform::ChangedStateException^) {
            continue;
        }
    } while (actual > expected);
    pairs.resize(actual);
    return pairs;
}

// override_or_fallback_t unchanged

template<typename M,
    typename KeyOverride,
    typename ValueOverride>
struct inferred_runtime_map_traits
{
    using KVP = decltype(std::declval<M>()->First()->Current);

    using Key = override_or_fallback_t<
        KeyOverride,
        decltype(KVP()->Key)>;
    using Value = override_or_fallback_t<
        ValueOverride,
        decltype(KVP()->Value)>;
};

// inferred_map_traits, inferred_unordered_map_traits,
// clone_as_map, and clone_as_unordered_map unchanged

template<typename Key = void,
    typename Value = void,
    typename Compare = void,
    typename M,
    typename Traits = inferred_map_traits
    <M, Key, Value, Compare>>
    auto CloneMap(M const& m,
        typename Traits::Compare const& compare = {})
{
    return ref new Platform::Collections::Map<
        typename Traits::Key,
```

```
        typename Traits::Value,
        typename Traits::Compare>(
        clone_as_map<Key, Value, Compare>
        (m, compare));
}

template<typename Key = void,
    typename Value = void,
    typename Hash = void,
    typename KeyEqual = void,
    typename M,
    typename Traits = inferred_unordered_map_traits
    <M, Key, Value, Hash, KeyEqual>>
    auto CloneUnorderedMap(M const& m,
        typename Traits::Hash const& hash = {},
        typename Traits::KeyEqual const& equal = {})
{
    return ref new Platform::Collections::UnorderedMap<
        typename Traits::Key,
        typename Traits::Hash,
        typename Traits::KeyEqual>(
        clone_as_unordered_map<Key, Value, Hash, KeyEqual>
        (m, hash, equal));
}
```

Hm, I thought this was going to introduce new wrinkles into the story, but I was pleasantly surprised: Everything translated to C++/CX in a straightforward manner with no gotchas.