Windows Runtime observable collections don’t mix well with multithreading

devblogs.microsoft.com/oldnewthing/20220708-00

July 8, 2022

Raymond Chen

The Windows Runtime provides observable collections `IObservableVector<T>` and `IObservableMap<K, V>`. Observability adds `VectorChanged` and `MapChanged` events (respectively) to allow you to be called back when the underlying collection changes.

These notifications interact poorly with multi-threading: What happens if while the thread is processing the previous change, another thread tries to mutate the collection?

Different implementations of the observable collection interfaces behave differently.

C#’s observable collections came first. From reading the reference source, we see that mutation methods throw an `InvalidOperationException` if they are mutated while a Changed event handler is active. (More detailed discussion in this StackOverflow question.) It is apparent that this object was designed for single-threaded use: The reentrancy checks apply to the object as a whole, regardless of thread. Furthermore, the code doesn’t block reentrancy until it’s about to raise the Changed event. Here’s an abbreviated version:

```
protected override void InsertItem(int index, T item)
{
    CheckReentrancy();
    base.InsertItem(index, item);

    using (BlockReentrancy())
    {
        /* raise the CollectionChanged event */
    }
}
```

That creates this multithreaded race condition:

Thread 1	Thread 2
InsertItem	

CheckReentrancy (succeeds)	
base.InsertItem	
	InsertItem
	CheckReentrancy (succeeds)
	base.InsertItem
BlockReentrancy	
Raise the event	
Handler sees inconsistent collection	

Note that Thread 1's Changed event handler is called after the collection has been changed by Thread 2, so when it goes to look at the collection to find the item that got inserted, it may not actually be there because Thread 2 already changed the collection.

This makes sense because the C# ObservableCollection is explicitly not thread-safe: `IsSynchronized` always returns `false`.

The Windows Runtime ValueSet and PropertySet are also observable, and they follow roughly the same model as the C# observable collections they were patterned after: Modifications to the collection are disallowed when a change notification is active. The operation will fail with the exception `RO_E_CHANGE_NOTIFICATION_IN_PROGRESS`. The Windows Runtime collections do take a little extra care to avoid the "inconsistent collection" problem: The concurrent call from Thread 2 fails rather than passing the initial concurrency check. (Basically by moving the `BlockReentrancy` to the top of the function.)

Observable maps created by C++/WinRT follow yet another pattern: They do not block subsequent operations while the Changed event is being raised. This means that handlers in this case have to be prepared for the case that the collection's state can change out from under them.

Oh, and what about C++/CX? Easy: They simply don't support concurrency at all!

> The C++/CX collection types support the same thread safety guarantees that STL containers support.

The concurrency policy for STL containers is that concurrent reads are permitted, but no other operation can be concurrent with a write.

What does this all mean for you?

Limit your use of observable collections to single-threaded scenarios. Observable collections were originally created for UI data binding, which is single-threaded, and that’s why the observable collection pattern doesn’t extend well to multi-threaded scenarios. Furthermore, do not mutate the collection during the change notification.

If you cannot avoid using observable collections in multi-threaded scenarios, then you have to understand that it’s not going to be a great experience. We’ve found four patterns so far:

Implementation	**Concurrent write**	**Write during notification**
C#	Unprotected	Rejected
C++/CX	Undefined	Undefined
ValueSet / PropertySet	Allowed	Rejected
C++/WinRT single_threaded_…	Undefined	Undefined
C++/WinRT multi_threaded_…	Allowed	Allowed

In the bottom row, we see that the C++/WinRT multi-threaded collections allow a write during a change notification. This means that change notification handlers in the bottom row need to be prepared for the possibility that the collection changes while the change handler is running.

We’ll look some more at that bottom row next time.

Raymond Chen

Follow