Adding state to the update notification pattern, part 6

devblogs.microsoft.com/oldnewthing/20240424-00

April 24, 2024

Raymond Chen

Last time, we built a stateful but coalescing update notification using a change counter to identify which request is the latest one, but noted that it does unnecessary work. Let's see if we can avoid the unnecessary work.

We could add some early exits to abandon the work if we notice that we are no longer doing work on behalf of the most recent text change. It means that we have to switch the change counter variable to a `std::atomic` since we will be reading the variable from the background thread at the same time the UI thread may be modifying it.

```
class EditControl
{
    ⟦ ... existing class members ... ⟧

    std::atomic<unsigned> m_latestId;
};

winrt::fire_and_forget
EditControl::TextChanged(std::string text)
{
    auto lifetime = get_strong();

    auto id = m_latestId.fetch_add(1, std::memory_order_relaxed);

    co_await winrt::resume_background();

    if (!IsLatestId(id))) co_return;

    std::vector<std::string> matches;
    for (auto&& candidate : FindCandidates(text)) {
        if (candidate.Verify()) {
            matches.push_back(candidate.Text());
        }
        if (!IsLatestId(id))) co_return;
    }

    co_await winrt::resume_foreground(Dispatcher());

    if (!IsLatestId(id))) co_return;

    SetAutocomplete(matches);
}

bool EditControl::IsLatestId(unsigned id)
{
    return id == m_latestId.load(std::memory_order_relaxed);
}
```

The background worker periodically checks whether its work has been discarded and abandons its efforts if so.

We'll wrap up this series by comparing the two solutions.