Corrections to Patterns for using the InitOnce functions

devblogs.microsoft.com/oldnewthing/20110420-00

April 20, 2011

Raymond Chen

Adam Rosenfield pointed out that it is not possible to fail an asynchronous initialization; if you pass `INIT_ONCE_INIT_FAILED` when completing an asynchronous initialization, the function fails with `ERROR_INVALID_PARAMETER`. (Serves me right for writing an article the night before it goes up.) A more correct version is therefore

```
ITEMCONTROLLER *SingletonManager::Lookup(DWORD dwId)
{
 ... same as before until we reach the "singleton constructor pattern"
 void *pv = NULL;
 BOOL fPending;
 if (!InitOnceBeginInitialize(&m_rgio[i], INIT_ONCE_ASYNC,
                              &fPending, &pv)) return NULL;
 if (fPending) {
  ITEMCONTROLLER *pic = m_rgsi[i].pfnCreateController();
  if (!pic) return NULL;
  if (InitOnceComplete(&m_rgio[i], INIT_ONCE_ASYNC, pic)) {
   pv = pic;
  } else {
   // lost the race - discard ours and retrieve the winner
   delete pic;
   InitOnceBeginInitialize(&m_rgio[i], INIT_ONCE_CHECK_ONLY,
                           X&fPending, &pv);
  }
 }
 return static_cast<ITEMCONTROLLER *>(pv);
}
```

In other words, the pattern is as follows:

- Call `InitOnceBeginInitialize` in async mode.
- If it returns `fPending == FALSE`, then initialization has already been performed and you can go ahead and use the result passed back in the final parameter.

- Otherwise, initialization is pending. Do your initialization, but remember that since this is a lock-free algorithm, there can be many threads trying to initialize simultaneously, so you have to be careful how you manipulate global state. This pattern works best if initialization takes the form of creating a new object (because that means multiple threads performining initialization are each creating independent objects).
- If initialization fails, then abandon the operation.
- Call `InitOnceComplete` with the result of your initialization.
- If `InitOnceComplete` succeeds, then you won the initialization race, and you're done.
- If `InitOnceComplete` fails, then you lost the initialization race and should clean up your failed initialization. In that case, you should call `InitOnceBeginInitialize` one last time to get the answer from the winner.

While I'm here, I may as well answer the exercises.

Exercise: Instead of calling `InitOnceComplete` with `INIT_ONCE_INIT_FAILED`, what happens if the function simply returns without ever completing the init-once?

Answer: The `INIT_ONCE` structure is left in an *asynchronous initialization pending* state. This is fine, because the next attempt to initialize will simply join the race. (And it will win since we already quit the race!)

Exercise: What happens if two threads try to perform asynchronous initialization and the first one to complete fails?

Answer: If two threads both begin initialization and the first one to come to a result concludes that the initialization fails, then it will abandon the initialization. The second thread will then come to its own conclusion. If that conclusion is also failure, then it too will abandon the initialization as well. If that conclusion is that initialization was successful, then its completion will succeed and the `INIT_ONCE` will enter the *initialized* state.

Exercise: Combine the results of the first two exercises and draw a conclusion.

Answer: It is fine to abandon a failed initialization (and indeed, given what we learned above, it is indeed mandatory).

There is a documentation update coming soon to clarify that you cannot combine `INIT_ONCE_ASYNC` and `INIT_ONCE_INIT_FAILED`.

Raymond Chen

Follow