Another pattern for using the InitOnce functions

devblogs.microsoft.com/oldnewthing/20161222-00

December 22, 2016

Raymond Chen

In my survey of patterns for using the InitOnce functions, I omitted the synchronous two-phase initialization.

The synchronous two-phase initialization is similar to the simple callback-based version in that only one thread gets to attempt an initialization at a time. But instead of doing the initialization in a callback, you do the initialization inline.

As a refresher, here's how you do it using `InitOnceExecuteOnce`:

```
BOOL CALLBACK AllocateAndInitializeTheThing(
    PINIT_ONCE initOnce,
    PVOID parameter,
    PVOID *context)
{
    *context = new(std::nothrow) Thing();
    return *context != nullptr;
}

Thing *GetSingletonThing()
{
    static INIT_ONCE initOnce = INIT_ONCE_STATIC_INIT;
    void *result;
    if (InitOnceExecuteOnce(&initOnce,
                            AllocateAndInitializeTheThing,
                            nullptr, &result)) {
        return static_cast<Thing*>(result);
    }
    return nullptr;
}
```

To use `InitOnceBeginInitialize` in synchronous mode, you basically move the callback function inline:

```
Thing *GetSingletonThing()
{
    static INIT_ONCE initOnce = INIT_ONCE_STATIC_INIT;
    void *result;
    BOOL pending;
    if (InitOnceBeginInitialize(&initOnce, 0,
                                &pending, &result)) {
        if (pending) {
            // Try to initialize the thing.
            result = new(std::nothrow) Thing();

            InitOnceComplete(&initOnce,
                result ? 0 : INIT_ONCE_INIT_FAILED,
                result);
        }
        return static_cast<Thing*>(result);
    }
    return nullptr;
}
```

You start by calling `InitOnceBeginInitialize`, and the value stored in the `pending` parameter tells you whether you need to run the initialization. If it says that you need to initialize, then do your initialization and then report the result back by calling `InitOnce-Complete`, saying either `0` to mean that initialization succeeded, or `INIT_ONCE_INIT_FAILED` to say that it failed.

If a second thread tries to initialize while an initialization is already in progress, the initial request waits to see what the result of the existing initialization is. If the existing initialization eventually succeeds, then the second initialization is told, "It's all good. No need to initialize." If the existing initialization eventually fails, then the second initialization is told, "Not yet initialized. Why don't you give it a shot?"

In other words, `InitOnceExecuteOnce` acts like a wrapper that goes roughly like this:

```
BOOL InitOnceExecuteOnce(
    PINIT_ONCE initOnce,
    PINIT_ONCE_FN callback,
    void* parameter,
    void** context)
{
  BOOL pending;
  BOOL success = InitOnceBeginInitialize(
                          initOnce, 0, &pending, context)) {
  if (success) {
    if (pending) {
      success = callback(initOnce, parameter, context);
      InitOnceComplete(initOnce,
        success ? 0 : INIT_ONCE_INIT_FAILED, *context);
    }
  }
  return success;
}
```

Here's a comparison table:

	InitOnce-Execute-Once	**InitOnceBeginInitialize Synchronous mode**	**InitOnceBeginInitialize Asynchronous mode**
How initialized	Callback	Inline	Inline
Initialization parallelism	Serialized	Serialized	Parallel
Success reporting	Callback returns TRUE	InitOnceComplete(0)	InitOnce-Complete(INIT_ONCE_ASYN(
Failure reporting	Callback returns FALSE	InitOnce-Complete(INIT_ONCE_FAILED)	Do nothing

Raymond Chen

Follow