How to turn off the exception handler that COM "helpfully" wraps around your server

devblogs.microsoft.com/oldnewthing/20110120-00

January 20, 2011

Raymond Chen

Historically, COM placed a giant `try/except` around your server's methods. If your server encountered what would normally be an unhandled exception, the giant `try/except` would catch it and turn it into the error `RPC_E_SERVERFAULT`. It then marked the exception as handled, so that the server remained running, thereby "improving robustness by keeping the server running even when it encountered a problem."

Mind you, this was actually a disservice.

The fact that an unhandled exception occurred means that the server was *in an unexpected state*. By catching the exception and saying, "Don't worry, it's all good," you end up leaving a corrupted server running. For example:

```
HRESULT CServer::DoOneWork(...)
{
 CWork *pwork = m_listWorkPending.RemoveFirst();
 if (pwork) {
   pwork->UpdateTimeStamp();
   pwork->FrobTheWidget();
   pwork->ReversePolarity();
   pwork->UnfrobTheWidget();
   m_listWorkDone.Add(pwork);
 }
 return S_OK;
}
```

Suppose there's a bug somewhere that causes `pwork->ReversePolarity()` to crash. Maybe the problem is that the neutrons aren't flowing, so there's no polarity to reverse. Maybe the polarizer is not property initialized. Whatever, doesn't matter what the problem is, just assume there's a bug that prevents it from working.

With the global `try/except`, COM catches the exception and returns `RPC_E_SERVERFAULT` back to the caller. Your server remains up and running, ready for another request. Mind you, your server is also corrupted. The widget never got unfrobbed, the timestamp refers to work that never completed, and the `CWork` that you removed from the pending work list got leaked.

But, hey, your server stayed up.

A few hours later, the server starts returning `E_OUTOFMEMORY` errors (because of all the leaked work items), you get errors because there are too many outstanding frobs, and the client hangs because it's waiting for a completion notification on that work item that you lost track of. You debug the server to see why everything is so screwed up, but you can't find anything wrong. "I don't understand why we are leaking frobs. Every time we frob a widget, there's a call to unfrob right after it!"

You eventually throw up your hands in resignation. "I can't figure it out. There's no way we can be leaking frobs."

Even worse, the inconsistent object state can be a security hole. An attacker tricks you into reversing the polarity of a nonexistent neutron flow, which causes you to leave the widget frobbed by mistake. Bingo, frobbing a widget makes it temporarily exempt from unauthorized polarity changes, and now the bad guys can change the polarity at will. Now you have to chase a security vulnerability where widgets are being left frobbed, and you still can't find it.

Catching all exceptions and letting the process continue running assumes that a server can recover from an unexpected failure. But this is absurd. You already know that the server is unrecoverably toast: It crashed!

Much better is to let the server crash so that the crash dump can be captured *at the point of the failure*. Now you have a fighting chance of figuring out what's going on.

But how do you turn off that massive `try/except`? You didn't put it in your code; COM created it for you.

Enter `IGlobalOptions`: Set the `COMGLB_EXCEPTION_HANDLING` property to `COMGLB_EXCEPTION_DONOT_HANDLE`, which means "Please don't try to 'help' me by catching all exceptions. If a fatal exception occurs in my code, then go ahead and let the process crash." In Windows 7, you can ask for the even stronger `COMGLB_EXCEPTION_DONOT_HANDLE_ANY`, which means "Don't even try to catch 'nonfatal' exceptions."

Wait, what's a 'fatal' exception?

A 'fatal' exception, at least as COM interprets it, is an exception like `STATUS_ACCESS_VIOLATION` or `STATUS_ILLEGAL_INSTRUCTION`. (A complete list is in this sample Rpc exception filter.) On the other hand a 'nonfatal' exception is something like a C++ exception or a CLR exception. You probably want an unhandled C++ or CLR exception to crash your server, too; after all, it would have crashed your program if it weren't running as a server. Therefore, my personal recommendation is to use `COMGLB_EXCEPTION_DONOT_HANDLE_ANY` whenever possible.

"That's great, but why is the default behavior the dangerous 'silently swallow exceptions' mode?"

The COM folks have made numerous attempts to change the default from the dangerous mode to one of the safer modes, but the application compatibility consequences have always been too great. Turns out there are a lot of servers that actually rely on COM silently masking their exceptions.

But at least now you won't be one of them.

Raymond Chen

Follow