Do WCF / IIS timeouts need a rewrite?

Asked 14 years, 9 months ago Modified 12 years, 7 months ago

Viewed 1k times

2

Recently I have been doing a lot of work on the WCF front and specifically hosting in IIS (not self hosting).

Is it just me, or does anyone have issues fine tuning timeout values. I'll start by mentioning the sheer number of timeouts you need to fine tune right off the bat.

Take a look at the following binding endpoint values that all in some way or another relate to a timeout:

- closeTimeout="00:01:00"
- openTimeout="00:01:00"
- receiveTimeout="00:10:00"
- sendTimeout="00:01:00"
- maxBufferSize="999"
- maxBufferPoolSize="524288"
- maxReceivedMessageSize="999"
- readerQuotas maxDepth="32"
- readerQuotas maxStringContentLength="8192"
- readerQuotas maxArrayLength="16384"

- readerQuotas maxBytesPerRead="4096"
- readerQuotas maxNameTableCharCount="16384"

These are just client side endpoint configuration values, we're not even getting started yet, now to get the service running in IIS, it is also required to setup the server side bindings, which offer the same level of complexity as the client side.

Once that is done, it is also essential to configure IIS otherwise during long calls to WCF services you will end up with your main thread being aborted.

IIS needs keep-alives disabled, the App Pool also comes with a vast array of timeout values, App Pool, this is a detailed subject in its own right. Besides this there are around another 7 timeout values that need to be specifically fine tuned, otherwise expect your complex WCF calls to fail.

Excuse me, but does anyone else smell a rat?

My understanding is that essentially (most) of these timeout values exist because of trust issues. By trust, I mean "We don't trust the services do do what they are supposed to do in a reasonable amount of time". Every aspect of SOA reliable communication is distrusted up front, and because of this, it seems we need vast arrays of catch nets (timeout values) in place to ensure some degree of manageability. Lets face it, if we trusted systems to give a response in a timely manner, why would be need to set a timeout value?

The issue I have with all this is that frankly its back to front, what if the 5 systems I have in my application are generally trustful and usually do give timely responses. I'm frustrated that I will still need to go through the lengthy process of defining these borders because OOB, WCF/IIS hosting fails.

I've observed that WCF technology is hypocritical, during web casts and marketing presentations, it is usually always mentioned that WCF allows better abstraction from the implementation, allowing developers to more easily write and deploy and by "simply"defining endpoints are able to concentrate on the business logic, and less on the underlining architecture. I have found in practice nothing can be further from the truth. With WCF you need to be deeply into the details of how your service operates, and you need to manually fine tune a great deal, unlike in ASMX services where usually they deployed without much fuss, and only needed some IIS fine tuning, rarely even.

So the question I pose is: Is it just me or do any of you share the same frustrations and observations? All comments welcome!

.net **wcf** **iis**

edited May 7, 2012 at 0:08

community wiki
4 revs, 2 users 99%
JL.

1 You know - to that one guy who marked the question down, clearly MS didn't think it was a bad idea, hence why the fixed my frustrations in v4.... – JL. Mar 7, 2010 at 1:31

IIS hosting always timeout for me. I'm seriously thinking in moving all of my services to self-hosting... – Jader Dias Aug 5, 2010 at 14:31

1 Answer

Sorted by: Highest score (default)

2

Where to start?

Something you have to remember is that WCF servces are architected to be independent from the hosting application. You can host WCF services in command-line apps, WinForms apps, Windows Services, IIS, Silverlight, Win32 apps, MFC apps, etc., as per your scenarios and needs.

Thus, the WCF infrastructure and the hosting app infrastructure need to be independently controlled and configured.

In your case you've chosen to host your WCF services in IIS. This is a fine choice for a broad range of scenarios, but a poor choice in others. Why? Because **IIS is specifically built for a very targeted scenario:**

receiving and dispatching requests to short-lived worker processes and returning the responses to the caller.

That "short-lived" statement was deliberate: IIS is primarily used as a web server. As such is configured by default to act as a web server. Web servers are generally configured to respond to the caller as quickly as possible. A web server doesn't know whether a worker process spawned by a page request is "busy" or "hung" unless the worker process returns a response. Until a response is received, the web server can only assume that the worker process is "busy". If the worker process hasn't responded within a given (configurable) time period, then the web server makes the decision to terminate the worker process as it's probably hung.

So, in your scenario, you're using IIS to host an atypical app which has special requirements (i.e. the ability to execute long-running operations). If you, the app developer/admin knows that certain apps may not return for more than 10 mins, then you can dial-in the time-out period for the worker process that hosts your special-case app. That is a good thing. If you were not given this ability, you'd be screaming when a bug in your code or a slow-down in your DB tier brought the entire web server to its knees killing every web site hosted by that machine because NONE of your worker processes timed-out.

Much the same logic applies to the WCF config settings:

All the settings you note allow you to control the performance characteristics of your WCF services by modifying their config settings - without having to change a line of code and/or deploy new bits (assuming you configure your services via config files rather than controlling these settings through code).

If, as you seem to indicated, you have WCF services that perform long-running tasks, you might choose NOT to host them in IIS and host them in a Windows Service app instead. In which case, your hosting app would do little other than host your services and respond appropriately to start-up and shut-down notifications instead. So how would you control your services' ability to handle messages with VERY large payloads? How would you control the number of service instances to be simultaneously created? How would you control how long WCF waits for a new service instance to open and/or close?

Be glad that you're given the opportunity to change none/some/all of these and a number of other settings and control the performance, security, reliability and behavior of your services so very easily. It could be a lot worse - you might have absolutely no control over your services or their host and have to live with whatever performance characteristics you were given.

answered Jun 4, 2010 at 17:37

community wiki
Rich Turner