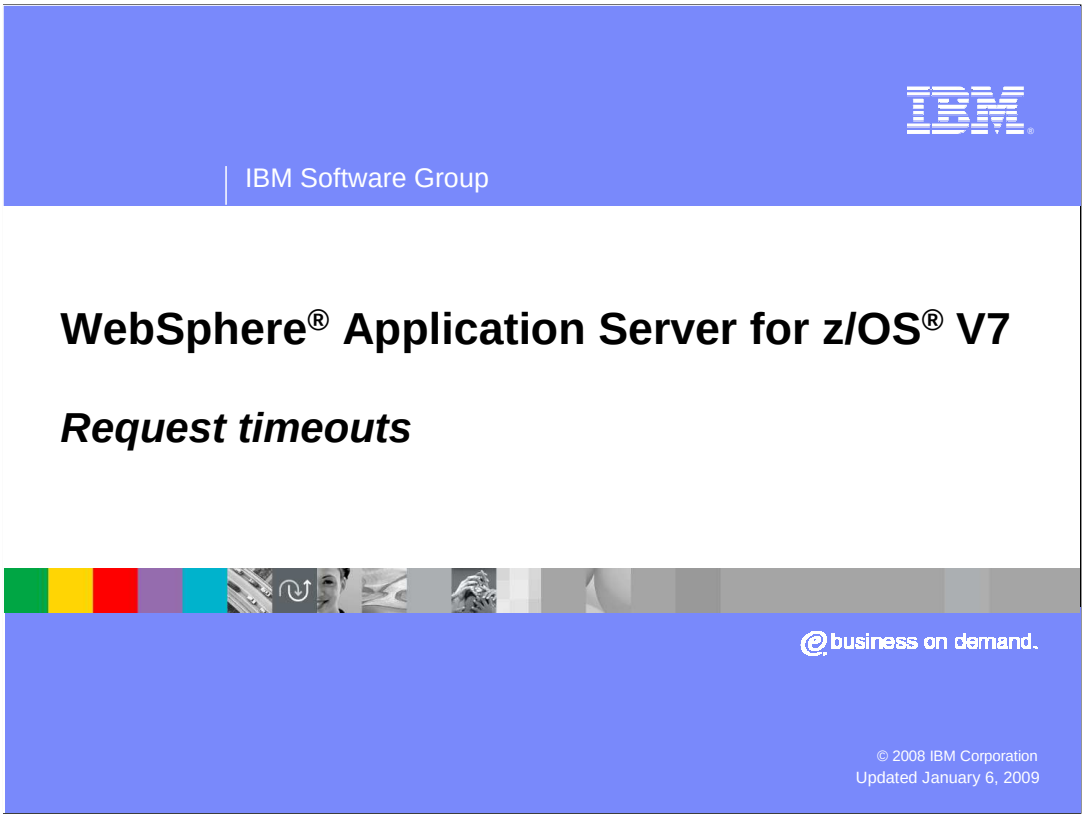


This presentation covers the request timeout functions available in WebSphere Application Server for z/OS.

Agenda

- Introduction
- Timeout configuration
- Timeout operational enhancements

In looking at timeout processing in WebSphere Application Server for z/OS, you will start with an introduction and some background information. Once you understand the need for timeouts, you will look at how to configure timeouts and then finally how to monitor threads for timeouts, if needed.

Timeout introduction

- Background (http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101233)
  - Applications have bugs, network connections hang, needed resources are unavailable…
- Architecture of WebSphere Application Server for z/OS
  - Server split into a controller region and one or more servant regions
  - Extra reliability and availability provided by identifying a misbehaving application request and, if necessary, ending it
  - Timeout values control amount of time allowed for request completion

In a perfect world, applications all behave, network connections never have problems and all the resources you need are always available. In the real world though, you know this is a pipe dream. Applications hang because of bugs introduced in their development, needed network connections go down, and database resources become unavailable. These problems can cause applications to take longer than expected, holding up other work. For instance, the application can be holding locks needed by other applications or might be in a loop, causing it to consume excessive amounts of processor time. Timeout processing allows the misbehaving servant to be dealt with. In the next few slides you will see the different possible actions available to deal with the problem, configurable by you.

Currently…

- Request takes ‘too long’ to process
  - ABEND the servant region, responding to all clients with a failure -- DEFAULT
    - Generates a dump of the servant
    - Penalizes work that was not having any problems
    - Throughput in the server can be affected waiting for servant restart
  - Allow the request to continue, responding to the client with a failure (SESSION)
    - Possibly allows the request to run to completion
    - If request is truly hung, the servant has one less thread for processing work
    - Problem can end up tying up multiple threads, eventually affecting throughput in the servant

Currently if a request is deemed to be taking ‘too long’ to process, the server assumes that the request must be hung and starts processing to rectify the situation. Depending on how you have the server configured, the servant where the request is running will either be terminated (`protocol_<http(s)/sip(s)>_timeout_output_recovery=SERVANT`) or the server will respond to the client and allow the request to continue processing (`protocol_<http(s)/sip(s)>_timeout_output_recovery=SESSION`).

In the case where the servant region is terminated, a dump of the servant can be generated and all work that was running in the servant is stopped as well. This option can end up penalizing work that was not having any problems. In addition, server throughput is affected while the dump is being taken and a new servant is started. This processing can take quite a while.

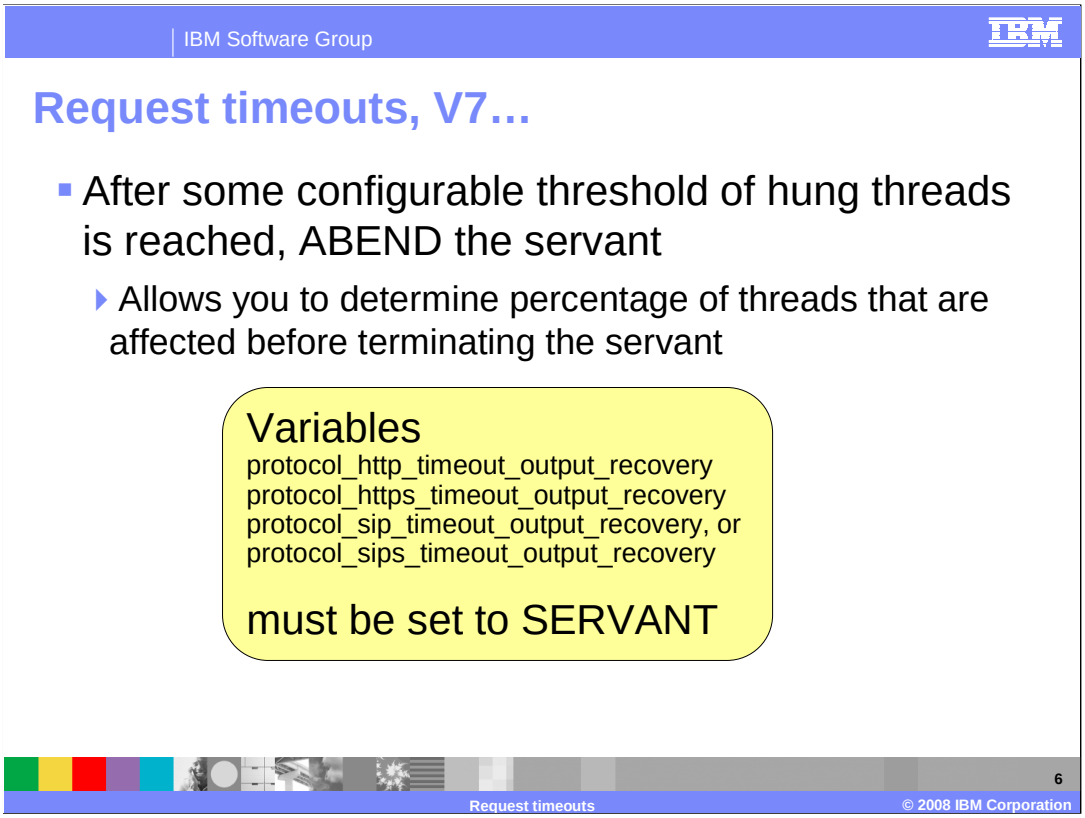
You can choose the option where the request is allowed to continue so that only the problem thread is affected when a hang is detected. This option assumes that there was an unusual event that caused the timeout and so the request will eventually complete successfully. If this assumption is a bad one, however, and the request is truly hung, the servant is left with one less thread for processing incoming work. In addition, by allowing the request to continue, deadlocks can occur because the request is holding locks or other resources. If this problem continues to happen on subsequent requests, multiple threads become tied up and the throughput of the servant is affected; possibly to the point where it has no threads available to process work.

Request timeouts, V7

- Encourage requests taking ‘too long’ to complete before considering them hung
  - Attempt to interrupt the work in progress (for instance, by using Java APIs to break out of java blocks)
    - If all goes as planned, the dispatched thread is freed and then either records the ‘event’ or turns it into an exception, or both
    - If attempt fails, servant eventually takes some doc and notifies the controller that the thread is hung
- Registry of ‘interruptible objects’ introduced
  - Blocking code can register so that it can be called to ‘unblock’ when thread becomes unresponsive
  - A java interruptible object always exists

In order to minimize the disruption to throughput, some new functionality has been added in Version 7.0 that tries to help the ‘hung’ dispatch thread to completion. If the servant notices that a request is hung, it will attempt to interrupt it so as to free the dispatched thread. If the servant is successful, it with either record the interrupt as an event or turn it into an exception of its own to be sent back to the client, or both. Before Version 7, depending on the configuration, the client will receive an exception and the request MAY eventually finish but it also may have been truly hung. If it was truly hung, this leaves one less thread to process work in the servant. The other option before Version 7.0 was to ABEND the servant. This affects throughput while a dump is taken and the servant is restarted.

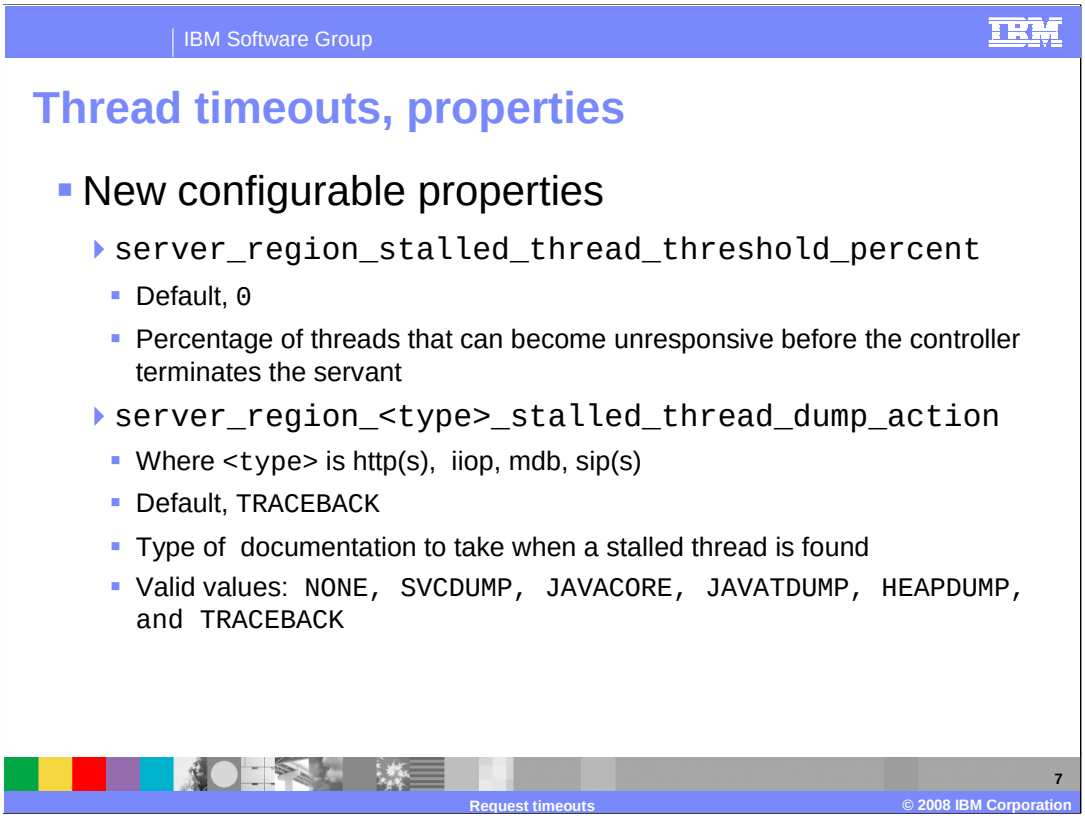
To allow the servant to try to interrupt a request, a new registry of ‘Interruptible Objects’ is introduced. Certain blocking code can register so that if too much time passes, the servant can call the ‘interruptible object’ in order for it to attempt to unblock the thread. A Java interruptible object will always be registered so the servant will try to have Java help interrupt the thread if all else fails.



Since it is possible that the servant may not be able to interrupt the work in progress successfully, there is still the possibility that the servant may need to be restarted. The difference in Version 7.0 though, is that now there is a configurable threshold of hung threads that needs to be reached before ABENDing the servant. This allows you to determine the percentage of threads that are considered hung, or non-working, before the servant is ABENDed and restarted.

If a thread that was reported to the controller as ‘hung’ finishes, the controller is notified of that so that it is no longer considered in the threshold determination.

In order to take advantage of this new function in version 7, the appropriate `protocol timeout output recovery variable must be set to ‘SERVANT’, which is the default.  If it is set to ‘SESSION’, a response is sent to the client and nothing else is done.`



To take advantage of this new request timeout function, there are a few new custom property variables you need to be aware of. The first one is the `server_region_stalled_thread_threshold_percent` variable. It tells the controller what percentage of threads should be considered hung before terminating the servant. If you leave the default value of zero, it will behave as it did in prior versions. It will either terminate the servant when it discovers that a thread is taking too long OR it will respond to the thread's client and allow the thread to keep running. The `server_region_<type>_stalled_thread_dump_action` variable tells the application server what documentation you want taken when a thread is unable to be freed and a notification of a stalled thread is sent to the controller.

Thread timeouts, properties…

- `server_region_request_cputimeused_limit`
  - Default, `0`
  - Specifies, in milliseconds, the amount of processor time that an application request can consume before the servant will attempt to take action
- `server_region_cputimeused_dump_action`
  - Default, TRACEBACK
  - Type of documentation to take when a request has consumed too much processor time
  - Valid values: `NONE, SVCDUMP, JAVACORE, JAVATDUMP,HEAPDUMP, and TRACEBACK`

To set:

Application servers > <serverName>

The last two custom properties you need to be aware of are `server_region_request_cputimeused_limit` and `server_region_cputimeused_dump_action`. These properties allow you to determine how much processor time you are willing to allow one request to consume. If the request exceeds the specified amount of time, UNIX Systems Services generates a signal that may or may not get delivered to the rogue request. The signal may not get delivered immediately if the thread has invoked a native PC routine, for instance. In that case, the signal will not get delivered until the PC routine returns to the thread. When and if the signal gets delivered, a BBOO0327 message is output, doc is gathered according to what you specified for the `server_region_cputimeused_dump_action` property and the controller is notified that a thread is hung. The default for the `server_region_cputimeused_dump_action` property is TRACEBACK. After delivery of the signal to the request, the WLM enclave token is quiesced, which results in the dispatch priority being lowered so that the thread should only be able to use the processor when the system is experiencing a light workload. If the hung-thread-threshold is hit, WebSphere will ABEND the servant as usual.

Thread timeouts, operational enhancements

- New command
  - `f server,display,threads,<parameters>`
    - Where parameters are:
      - `ALL`
      - `TIMEDOUT`
      - `REQUEST=<value>`
      - `ASID=<value>`
      - `AGE=<value>`
    - Can further qualify with
      - `SUMMARY (default, except for REQUEST=<value>)`
      - `DETAILS`
- Information also available by way of a new InterruptibleThreadInfrastructure MBean

There is a new command to display the dispatch threads that are currently active. The DISPLAY,THREADS command will display information about every dispatch thread in every servant region associated with the specified controller. By default, it will give you SUMMARY information but you can also specify that you want DETAILS. In the case of the REQUEST=<value> parameter, the default is DETAILS.

The information is also available by way of a new InterruptibleThreadInfrastructure MBean.

Thread timeouts, operational enhancements…

```
F S7SR01A,DISPLAY,THREADS,ALL
  BBOJ0111I:  REQUEST   ASID JW TO RE DISPATCH TIME
  BBOJ0112I: ffffe453 0X0041  Y  N  N 2008/08/23 18:38:12.391628
  BBOJ0112I: ffffe452 0X0041  N  N  N 2008/08/23 18:38:27.473191
  BBOJ0112I: ffffe454 0X0041  Y  N  N 2008/08/23 18:38:12.319306
  BBOJ0112I: ffffe451 0X003C  N  N  N 2008/08/23 18:38:27.485103
  BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,THREADS,ALL
```

Where JW = ‘request is in a Java Wait’

TO = ‘Timed Out’

RE = ‘Retry count Exceeded’

Here you see an example of the default dispatch thread display. There are four threads currently dispatched and two of them are currently in a Java Wait. If you want more information about a particular one, you can use the REQUEST=<value> option for that request. None of the requests shown have timed out as shown by the ‘TO’ heading. When the controller is informed that the attempt to ‘free the dispatch thread’ has failed, the ‘RE’ column is changed to ‘Y’. The last column, dispatch time, shows the time when the request arrived in the servant region to be dispatched. The time shown is local time if ras_time_local is set to 1, otherwise the time shown is GMT.

Thread timeouts, operational enhancements…

```
F S7SR01A,DISPLAY,THREADS,DETAILS
BBOJ0106I: REQUEST ffffe453 ASID 0X0041 TCB 0X008CAE88
BBOJ0119I: CONTROLLER RECEIVED REQUEST AT 2008/04/14 18:38:12.391478
BBOJ0120I: CONTROLLER QUEUED REQUEST TO WLM AT 2008/04/14 18:38:12.391522
BBOJ0107I: SERVANT DISPATCHED REQUEST AT 2008/04/14 18:38:12.391628
BBOJ0108I: JVM THD IS HUNG: ITI INACTIVE
BBOJ0110I: DETAILS FOR JVM INTERRUPTIBLE THREAD: Monitor ACTIVE
BBOJ0106I: REQUEST ffffe450 ASID 0X0041 TCB 0X008CA0B8
BBOJ0119I: CONTROLLER RECEIVED REQUEST AT 2008/04/14 18:42:27.170745
BBOJ0120I: CONTROLLER QUEUED REQUEST TO WLM AT 2008/04/14 18:42:27.170851
BBOJ0107I: SERVANT DISPATCHED REQUEST AT 2008/04/14 18:42:27.170899
BBOJ0108I: JVM THD IS NOT HUNG: ITI INACTIVE
BBOJ0110I: DETAILS FOR JVM INTERRUPTIBLE THREAD: Monitor ACTIVE
BBOJ0106I: REQUEST ffffe454 ASID 0X0041 TCB 0X008C9A48
BBOJ0119I: CONTROLLER RECEIVED REQUEST AT 2008/04/14 18:38:12.319285
BBOJ0120I: CONTROLLER QUEUED REQUEST TO WLM AT 2008/04/14 18:38:12.319302
BBOJ0107I: SERVANT DISPATCHED REQUEST AT 2008/04/14 18:38:12.319306
BBOJ0108I: JVM THD IS HUNG: ITI INACTIVE
BBOJ0110I: DETAILS FOR JVM INTERRUPTIBLE THREAD: Monitor ACTIVE
BBOJ0106I: REQUEST ffffe44f ASID 0X003C TCB 0X008CA1E0
BBOJ0119I: CONTROLLER RECEIVED REQUEST AT 2008/04/14 18:42:27.178812
BBOJ0120I: CONTROLLER QUEUED REQUEST TO WLM AT 2008/04/14 18:42:27.178956
BBOJ0107I: SERVANT DISPATCHED REQUEST AT 2008/04/14 18:42:27.178964
BBOJ0108I: JVM THD IS NOT HUNG: ITI INACTIVE
BBOJ0110I: DETAILS FOR JVM INTERRUPTIBLE THREAD: Monitor ACTIVE
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,THREADS,DETAILS
```

Just to get an idea of what the DETAILS option shows, here are the same four dispatch threads a bit later. Notice that two of the threads are now considered hung. The ‘ITI Inactive’ message refers to the ‘Interruptible Thread Infrastructure’ and indicates that it is not presently trying to interrupt the hung request. The BBOJ0110I message is printed out for each registered ‘interruptible object’. Note each of these just shows the default JVM interruptible object. The ‘Monitor Active’ tells you that the interruptible object was successfully registered. Other interruptible objects can provide different information about what blocking activity they are registered to handle.

Summary

- Threads sometimes take too long to process
  - Timeout configuration can help
  - Thread monitoring available

In summary, unforeseen circumstances can cause problems in a production environment where threads can take too long to process the work they were dispatched to do. Timeouts are helpful in guarding your system against these unforeseen circumstances. Timeouts allow actions to be taken in order to free threads that may be hung. This presentation showed you the different ways to configure your system in order to best deal with problems that occur.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASv7zOS_ThreadTimeouts.ppt

This module is also available in PDF format at: ../WASv7zOS_ThreadTimeouts.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere z/OS

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

JVM and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.