Lab 7. Troubleshooting in Tomcat

Every day, IT administrators face new problems with servers in the production environment. Administrators have to troubleshoot these issues to make sure that the applications work perfectly.

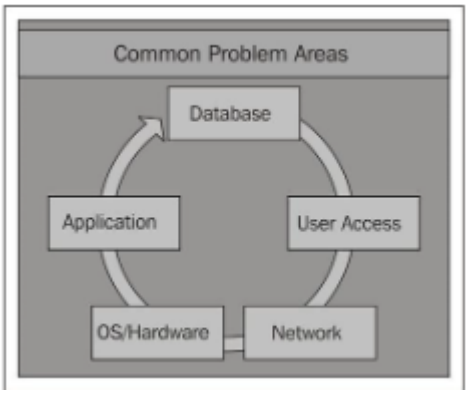
Troubleshooting is an art of solving critical issues in the environment. It comes with experience and the number of issues you have come across in your career. But there is a set of rules for fixing the issues. We will discuss the real-time issues, which may occur in the production environment. We will also discuss tips and tricks for resolving issues.

In this lab, we will discuss:

- Common issues
- Third-party tools for thread dump analysis
- Tomcat specific issues related to the OS, JVM, and database
- How to troubleshoot a problem
- Best practices for the production environment

Common problem areas for web administrators

Web administrators always find issues with applications, not due to server failure of the Tomcat server, but because applications start malfunctioning due to other components as well. The following figure shows different components for a typical middleware environment:



Let's briefly discuss the issues encountered by web administrators in real-time production support:

- **Application:** These issues occur when an application doesn't work correctly due to reasons such as class loader conflicts, application deployment conflicts, configuration parameters missing, and so on.
- **Database:** Database issues are very critical for the web administrator. It is very difficult to find the issues related to the DB. Some of them are; JNDI not found, broken pipe errors, and so on.
- **User Access:** Access issues can occur due to database or application mis-configuration. Some examples of these issues are; users are not able to access the application, login page doesn't appear, access denied, and so on.
- **Network:** This plays a vital role in the IT infrastructure. If the connectivity between the servers goes down, then the communication between the servers also goes down, and we face an interruption of services.
- **OS/Hardware:** The OS/hardware creates the lower layer where the application layer resides. If there are any issues with reference to the OS/hardware, it will affect the services of the Tomcat server.

How to troubleshoot a problem

We cannot troubleshoot any issue by just referring to the user comments or problem statement. In order to troubleshoot the issue, we have to narrow down the problem to its root level and fix the issue.

Many web administrators always ask this question; how do we know that the system has a particular problem?

The solution to these issues will be found, if you dig the problem in the correct path. Secondly, if you come across a number of problems in your career, then you can correlate them and solve the problem. If you ask me, practically it's impossible to teach troubleshooting, as it comes from your own experience and your interest to solve the problem. Here, we discuss one of the common problems,[*application slowness*], that occurs in every environment and the web administrator has to face this problem in his/her career.

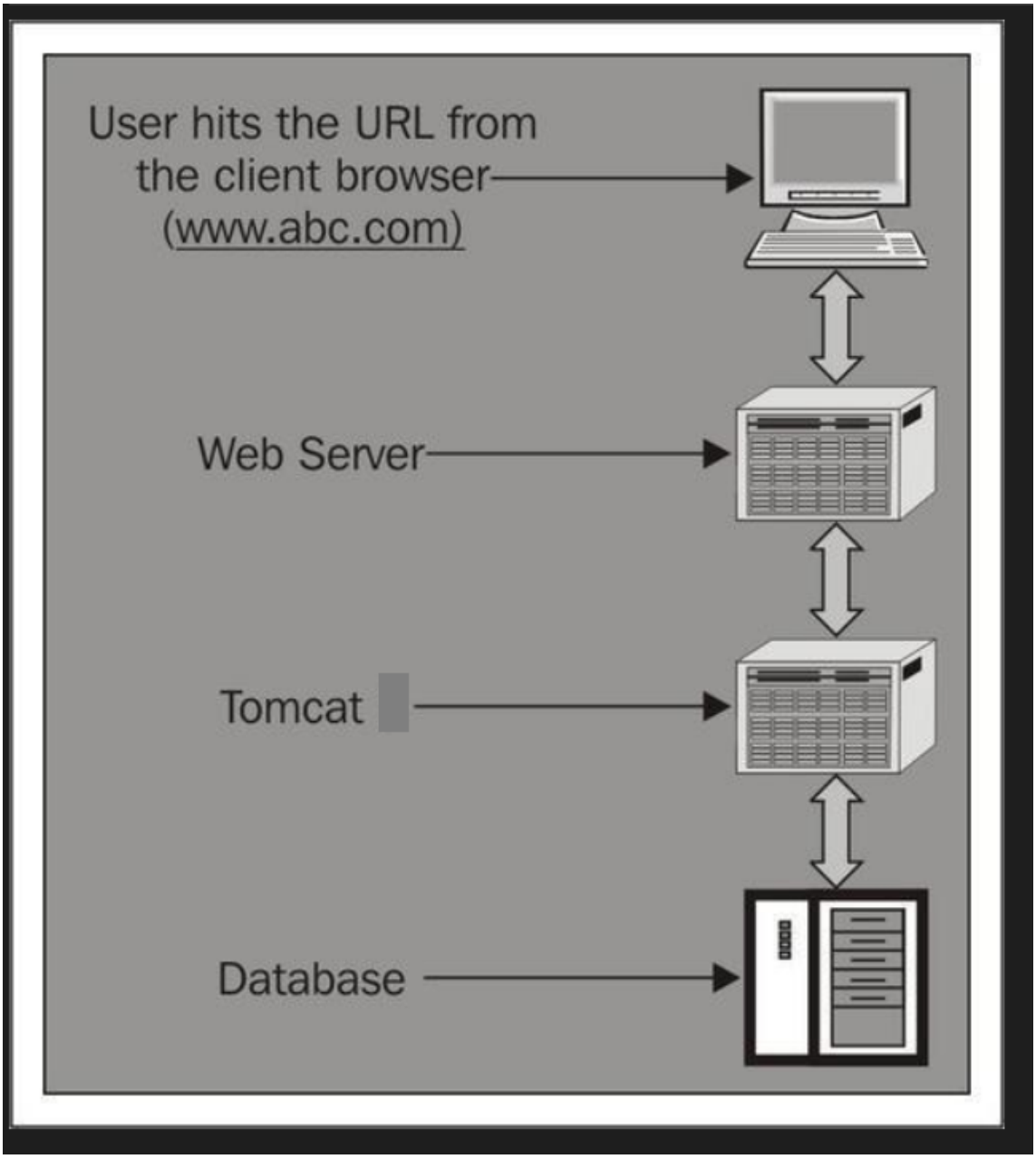
Slowness issue in applications

Let's take a real-time situation where users complain about the performance of the application. The application comprises of an enterprise setup, which is a combination of the Apache HTTP server as a frontend, Tomcat 8 is used as a servlet container, and the Oracle database running as a backend database server.

Issue:

Let's discuss one of the common issues of the middleware application, which make it very difficult for the administrator to solve. This issue is called[*slowness of application*], where users complain that the application is running slow. It's a very critical problem from the administrator's point of view, as slowness can be caused by any component of the web application, such as the OS, DB, web server, network, and so on.

Until and unless we find out which particular component is causing the problem, the slowness will persist and from the user's point of view, the application will not run in a stable manner. The following figure shows the typical web infrastructure request flow for a web application:



How to solve slowness issues in Tomcat 8

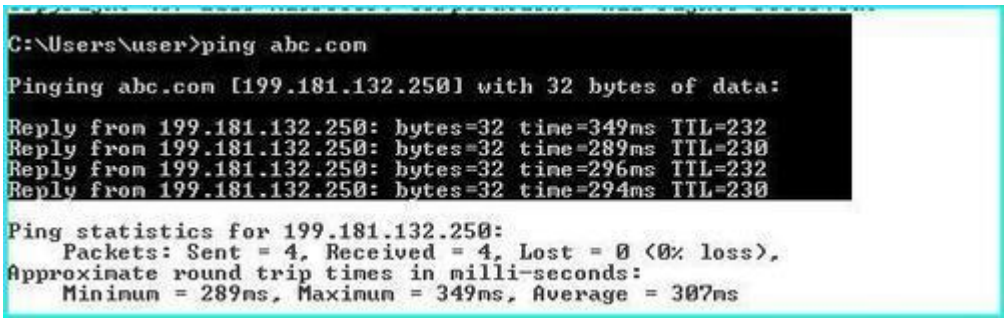
Slowness in the application can be caused by any component, so it is best practice to start troubleshooting from the user end.

User end troubleshooting

Perform the following steps to troubleshoot:

1. Try to access the application from the user's browser and check how much time it takes to load the application page.
2. Check the ping response of the server from the user side, for example, abc.com, using the command ping. If you get an appropriate response, it means the connectivity for the application server and user machine is working fine.

```
ping abc.com
```



- The previous screenshot indicates the `ping` response for `abc.com`. There are some important points we have to keep in mind during ping status monitoring, which are mentioned as follows:
- The packet sent and received should have an equal count. In the

```
previous screenshot we can see it\'s` 4`. If the count
is less, it means that there is some issue within the network.
```

- There should be no packet loss. Also, the average response time

```
should not be high.
```

The previous screenshot shows the `ping` response for the server working appropriately. That means there are no issues from the user end in terms of the system and network.

Web server troubleshooting

Once we know that there are no issues at the user end, we will move to the next level in the application, that is web server. Now, we have to dig down in the server to check if there are any issues.

Web server issues are more often related to the load of the server, user threads, or mounting problems. Let us see how to solve the issue.

1. Check whether the web server process is running or not. If it is running, check how many processes are running by using the following command. This command will show the number of processes and their status.

```
ps -aef |grep httpd
```

- The previous command shows the number of the processes running

```
for the Apache httpd server. If the processes are greater than
50, it means that there is some issue with the web server such
as a high CPU utilization, high user traffic, high disk I/O, and
so on.
```

2. Then, check the CPU utilization and memory status of the system to see if any Apache processes are consuming a high CPU usage by using the following command:

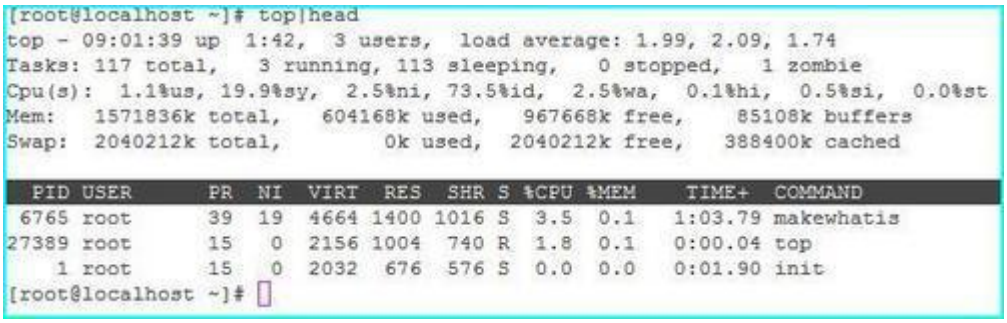
```
top|head
```

- The previous command will display the process which consumes the

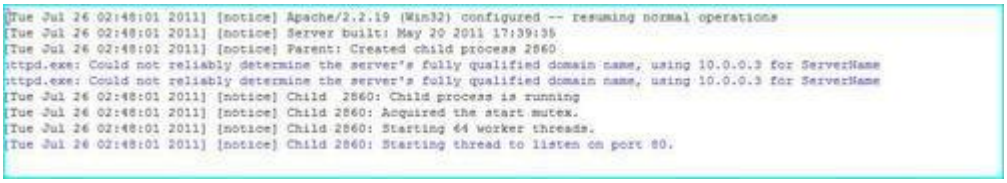
```
highest CPU usage and load average of the machine. The following
screenshot shows the output of the previous command. If the load
average is high or Apache process has a high CPU utilization,
then it is one of the reasons for slowness in the application,
otherwise we can proceed to the next level.
```

Note

```
In such cases, as mentioned earlier, you have to kill all Apache
processes and then recycle the Apache instance.
```



3. The next step is to check the Apache logs and search for errors in the error and access logs. The following screenshot shows the system has started successfully:




```
```
httpd.exe: Could not reliably determine the server's fully qualified domain name,
using 10.0.0.3 for ServerName.
```
```

- The previous message is a notification message (info) in

```
`apache error_log`. The log in the previous screenshot
shows that \" the Apache HTTP server could not find a fully
qualified domain\". This means that in the
`httpd.conf`, we have missed defining the server name
with a fully-qualified domain, for example, we have defined the
localhost as the server name; instead of that, we have to define
```

```
`localhost@localdomain.com`.

Also, there are two commands which are useful for searching the
error in the logs. They are as follows:
```

```
```
tail -f log file |grep ERROR
```
```

- The previous command is used when you want to search the error

```
in the logs.
```

```
```
grep " 500 " access_log
```
```

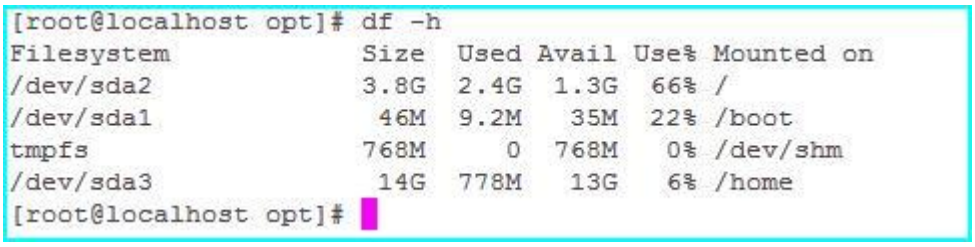
- The previous command is used to search error codes in the logs.

Note

```
In case logs are not generated for Apache, it may be due to the hard
drive running out of space.
```

4. One of the major reasons for the hard drive running out of space on the server mount, where application logs are mounted, is improper log rotation. Use the df command to check the mount space, where df = disk free and switch -h = human readable. The syntax to use the df command is as follows and the output is shown in the following screenshot:

```
df -h
```



Note

```
If any mount is running greater than 95 percent, then reduce the
disk utilization, otherwise the system may cause a disruption of
services.
```

If we don't find any error in the previously mentioned components, than we can conclude that there are no issues with the web server.

Tomcat 8 troubleshooting

In Java-based applications, slowness is caused due to many issues. Some of them are due to the JVM memory, improper application deployment, incorrect DB configuration, and so on. Let's discuss some basic troubleshooting steps for Tomcat 8:

1. Check the Java processes for Tomcat and the load average for the instance machine:

```
ps -ef |grep java
```

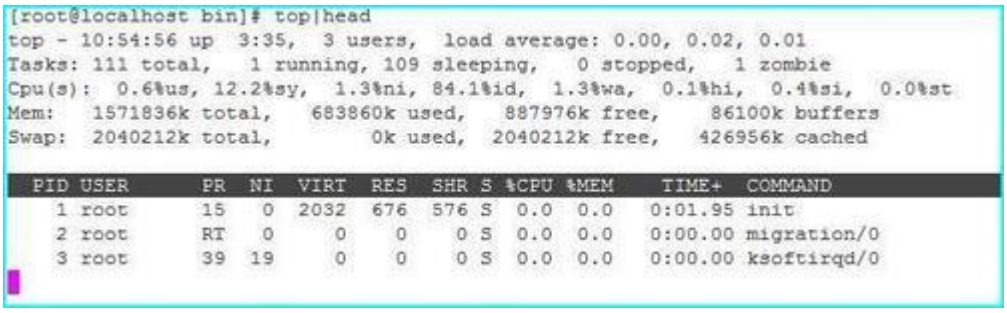
- The previous screenshot shows the Java processes running in the

```
machine. The previous command checks all the Java processes
running in the system and the load average for the Tomcat
instance. The load average gives us some important clues. In
case you find the load average is very high, then check which
process has a high CPU usage and find out the reason for using a
high CPU. Also, it shows the RAM and Swap usage.
```

```
The following screenshot shows the output of the
`head` command on the Tomcat server:



```
top|head
```
```



- The `head` command displays the content from the

```
first line of a file or output. It is very frequently used with
`-n` switch where `n=` the number of lines
to display. By default, it displays 10 lines if `-n`
is not used.
```

2. Then check the Tomcat logs which can be found in TOMCAT_HOME/logs, and search for the exception in the log files, mainly in catalina.out, localhost.yyyy-mm-dd.log using the following command:

```
grep INFO catalina.out
```

- The previous screenshot shows the Tomcat startup in the logs. If

```
there are any errors in the logs, they can be checked using the
following command:
```

```
grep ERROR catalina.out
```

Troubleshooting at the database level

As a web administrator, you don't have access to the database servers. But a web administrator can connect to the DB server externally, without logging into a physical machine, as the administrator has the connection string (credentials for accessing the database). For example, you can do the telnet on the port where the DB server is running, and check whether the services are running or not.

Telnet DB server IP port

If the telnet is successful, then you can verify the following processes:

- **Number of database connections:** We can always ask our DBA to check the number of connections on the database. If the connections count is high, then we can work with the DBA to reduce the connections on the server.
- **SQL query optimization:** We can check with the DBA to see which queries consume more time to execute in the database and ask our developers to optimize the query. This really helps in improving the performance of the application.
- **Load balancing database across multiple servers:** Another important point which may cause slowness in the application is the load balancing of the database across multiple servers. If the load balancing is not configured correctly, then it may cause slowness in the application. If there is a delay in the network between the two database servers, then sync may not happen appropriately.

JVM analysis in the Tomcat instance

There are some chances where the JVM is over utilized in the application. To view the memory allocation for the JVM instance, you can use the command-line utility, `jmap`. It's a Java utility, which determines the entire memory allocation of the Tomcat instance.

```
[root@localhost logs]# jmap -heap "TOMCAT INSTANCE PID "
```

Let us discuss how the previous command performs. The `jmap` command internally collects the JVM memory details, `-heap` is the switch that tells `jmap` to collect and display the heap memory footprint, `TOMCAT INSTANCE PID` is the process ID of the Tomcat instance for which process `jmap` has to fetch the memory details.

```
[root@localhost logs]# jmap -heap 10638
```

The following screenshot shows the output of the `jmap` command for the previous process ID:

Note

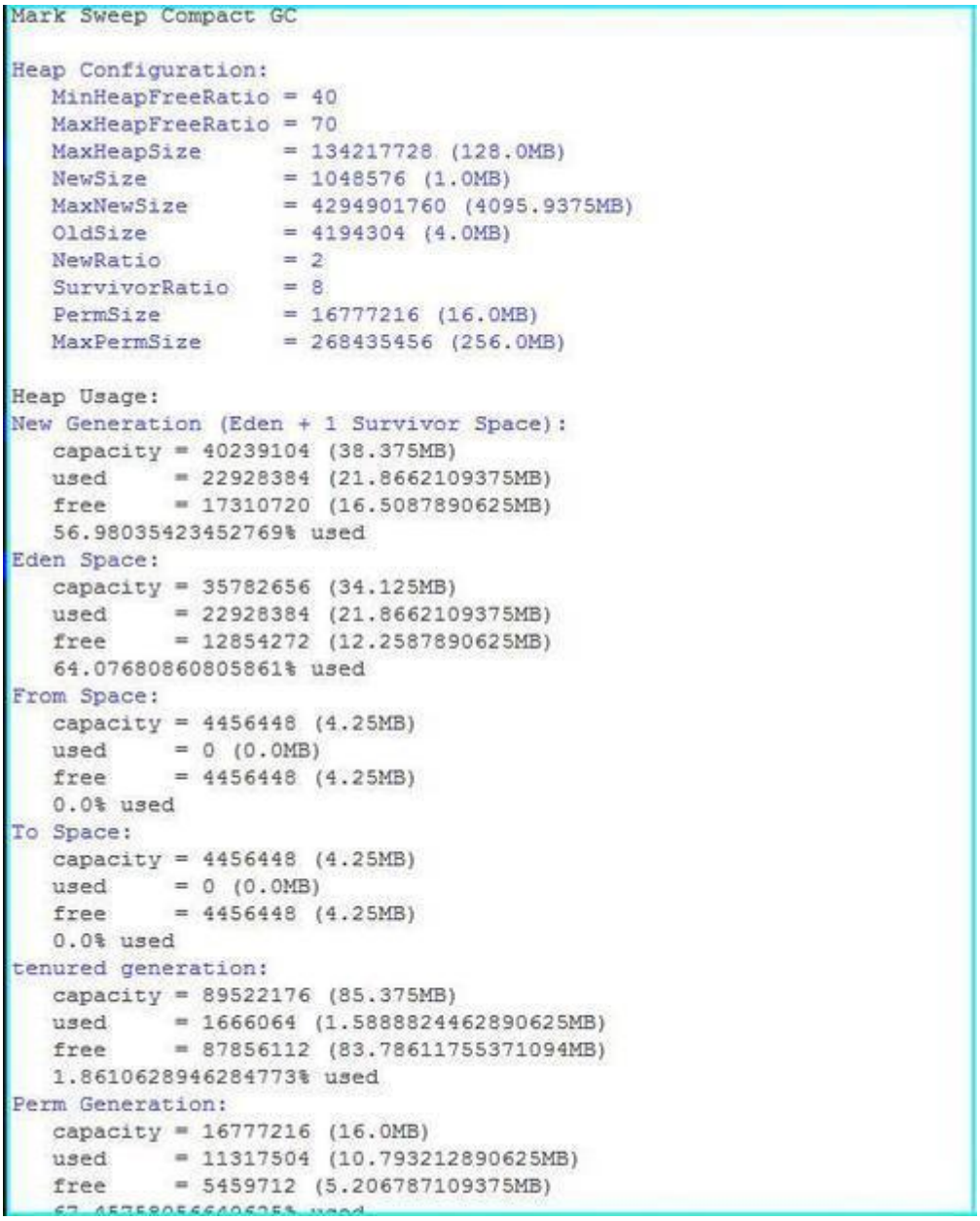
How to find the process ID

We can find the process ID using the following command:

```
ps -ef |grep "tomcat instance name " |awk -F" " '{print $2}'|head -1
```

This command can be described as, `ps -ef |grep "tomcat instance name "` will find all the processes running for the Tomcat instance. `awk -F" " '{print $2}'` awk prints the process ID of a particular process and `head -1` will display the first process ID.

The `jmap` command is present in `JAVA_HOME/bin` and if you set the `JAVA _HOME/bin` in the path, then you can execute the command from anywhere.



```
Mark Sweep Compact GC

Heap Configuration:
   MinHeapFreeRatio = 40
   MaxHeapFreeRatio = 70
   MaxHeapSize      = 134217728 (128.0MB)
   NewSize          = 1048576 (1.0MB)
   MaxNewSize       = 4294901760 (4095.9375MB)
   OldSize          = 4194304 (4.0MB)
   NewRatio         = 2
   SurvivorRatio    = 8
   PermSize         = 16777216 (16.0MB)
   MaxPermSize      = 268435456 (256.0MB)

Heap Usage:
New Generation (Eden + 1 Survivor Space):
   capacity = 40239104 (38.375MB)
   used     = 22928384 (21.8662109375MB)
   free     = 17310720 (16.5087890625MB)
   56.98035423452769% used
Eden Space:
   capacity = 35782656 (34.125MB)
   used     = 22928384 (21.8662109375MB)
   free     = 12854272 (12.2587890625MB)
   64.07680860805861% used
From Space:
   capacity = 4456448 (4.25MB)
   used     = 0 (0.0MB)
   free     = 4456448 (4.25MB)
   0.0% used
To Space:
   capacity = 4456448 (4.25MB)
   used     = 0 (0.0MB)
   free     = 4456448 (4.25MB)
   0.0% used
tenured generation:
   capacity = 89522176 (85.375MB)
   used     = 1666064 (1.5888824462890625MB)
   free     = 87856112 (83.78611755371094MB)
   1.8610628946284773% used
Perm Generation:
   capacity = 16777216 (16.0MB)
   used     = 11317504 (10.793212890625MB)
   free     = 5459712 (5.206787109375MB)
   67.45758056640625% used
```

The previous utility gives the entire footprint of the JVM memory and its allocation for the Tomcat instance. The JVM memory comprises of the following components:

- Heap configuration
- Heap usage
- From space
- To space
- Tenured generation
- Perm generation
- Eden space

Out of memory issues such as perm generation and max heap are very commonly known issues in the production environment. Check the memory to see whether any of the previous components are utilizing more than 95 percent. If so, then we have to increase the respective parameter.

Now it comes to the place where we can determine which JVM component is creating the issue for the Tomcat instance. If the memory is working fine, then it is time to generate a thread dump to drill the application-level issue.

How to obtain a thread dump in Tomcat 8

The thread dump is a way through which we can determine the application-level thread status for any Java process. There are many ways to obtain a thread dump in Tomcat; here we will discuss two different ways which are widely used in the IT environment.

Thread dump using Kill command

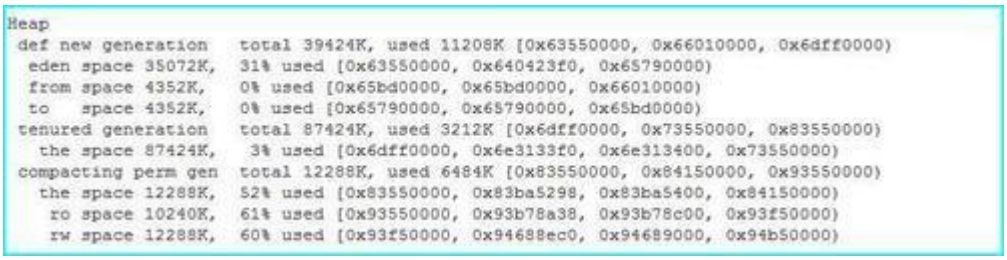
This command generates and redirects the thread dump in `catalina.out` log. But, the limitation to this command is it works in a non-DOS environment such as Linux, Unix, and so on.

```
Kill -3 java process id
```

For example:

```
Kill -3 10638
```

The previous screenshot shows the output of the thread dump command in `catalina.out` logs. We can see that the highlighted section shows the `http-bio-8080- Acceptor` thread status, which is currently in a runnable state, which means that the thread is alive and performing its functionality for the application.



Once the thread generation is complete, it then collects the memory dump for the Java processes. The previous screenshot shows the memory status at the time of the thread dump. This memory dump gives us the complete footprint of the memory used.

Thread dump using jstack

There is another way of generating a thread dump, that is using the Java command-line utility called `jstack`, which comes with JDK 1.5 or later versions. `jstack` prints the Java stack thread for a Java process. This utility is very useful in a production environment, where we have not redirected the thread output in the server logs. The major advantage of this utility is that it can be used with any J2EE server. There are some switches which are commonly used with the `jstack` command, as mentioned in the following table:

Options	Description
`-f`	Generates a Java stack forcefully. Majorly used when the process is in the hang state
`-l`	Long listing (displays the additional information on locks)
`-m`	Mixed mode Java stack generation

The following command syntax generates a Java stack for a Java process and redirects the output in a text file:

```
jstack -F Pid > threaddump.txt
```

For example:

```
jstack -F 10638 > threaddump.txt
```

JStack on Windows: On the Windows system, only one switch will work, and that is, `jstack [-l] pid`.

How to analyze the thread dump for Tomcat instance

The thread-dump analysis is tricky to understand. It gives a deep insight of application-related issues to the IT administrators. The following approach is applicable to do the thread-dump analysis. The steps are mentioned as follows:

1. Obtain the thread dumps six times for the Java process ID with an interval of 10 seconds, using the command kill -3 or jstack.
2. Then compare all the six thread dumps to find the long running threads.
3. Find all the threads in the stuck state and try to find the reason for all the stuck threads for the application and server-level threads.

Note

```
If the stuck thread is at the application level, then the issue is
related to the application code and if the thread is stuck at the
server level, then it may be a server or application-level issue.
```

Note

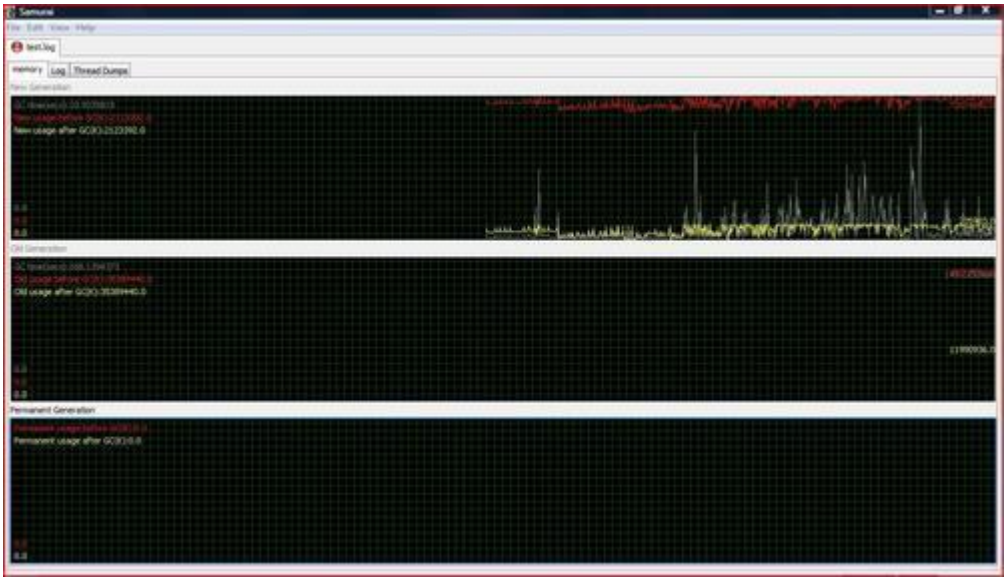
For Samurai, check the link http://yusuke.homeip.net/samurai/en/index.html.

Thread dump analysis using Samurai

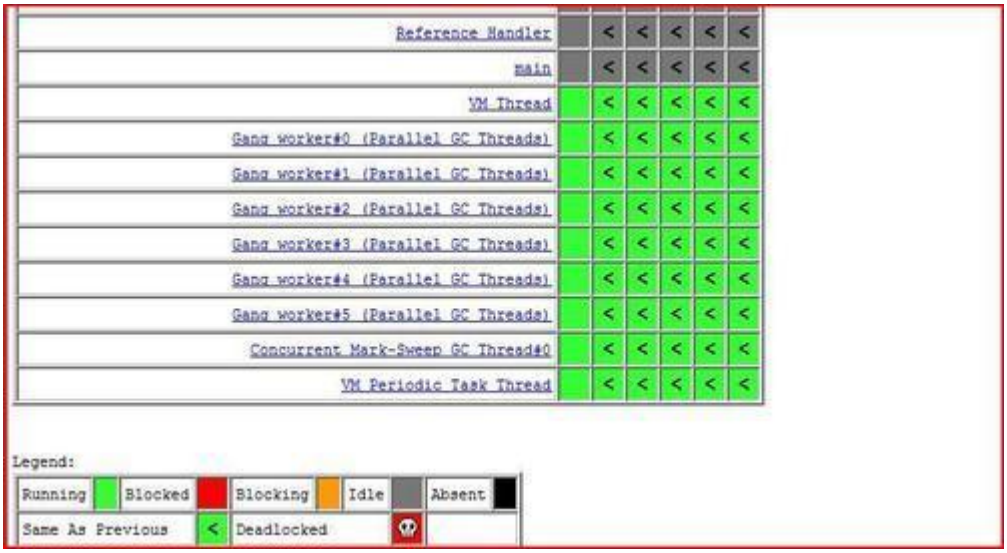
In the previous section,[*How to analyze the thread dump for a Tomcat instance*], we have discussed the various steps for the thread-dump analysis. Let's do a real-time analysis using a Samurai tool.

It's a GUI-based tool, which has the capability to segregate the thread dump and verbose GC from the log file and display it on a user-friendly screen. Let's start the analysis process:

1. Run the Samurai thread dump analyzer online using the link http://yusuke.homeip.net/samurai/en/index.html.
2. Once the Samurai console is open, upload the logs to the Samurai tool.
3. The Samurai tool internally separates the logs and visualizes the thread dump and memory dashboard. The following screenshot shows the GC verbose utilization displayed by Samurai:

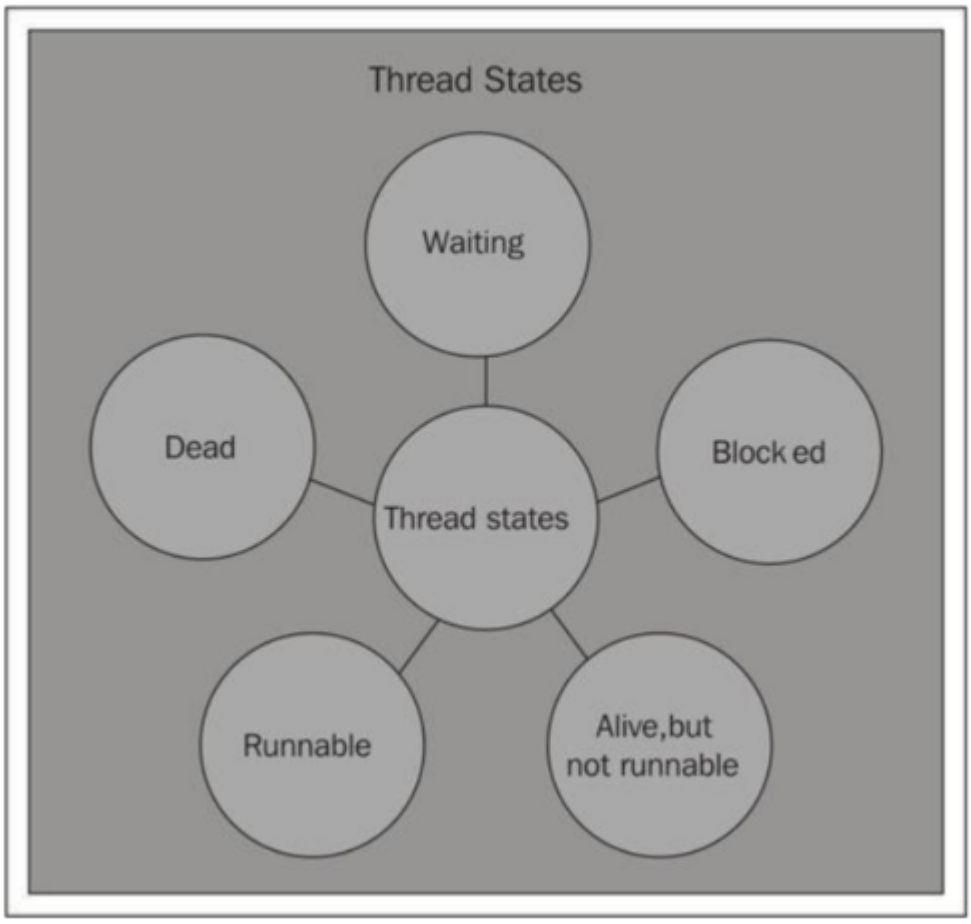


4. Now, click on the Thread Dumps tab, it will display the graphical status of the thread dump. The following screenshot shows the thread status. Also, on the left-hand corner, we can find the description of the symbol used for the thread dumps:



5. Now that we have the complete information, compare long threads manually and find the issue.

Thread states can be classified into five types. The following figure shows the different thread states:



- **Runnable:** A thread goes in the runnable state once the `start()` method is invoked.

- **Waiting:** A thread goes in the waiting state when it waits for a resource to get allocated or another thread to execute its function.
- **Blocked:** A thread goes in the blocked state when it waits for the monitor lock.
- **Alive but not runnable:** A thread in this state is still alive but not runnable, it might return to the runnable state later, if that method is invoked again.
- **Dead:** A thread goes in the dead state once its operation is complete, or when its operation is terminated abnormally. If any thread comes to this state, it cannot ever run again. It's also called a deadlock state.

Dead threads are critical threads for the instance, as they cause slowness in the application. They can be released as part of the garbage collection and the system will generate new threads causing memory leaks.

Till now we have discussed the troubleshooting steps performed for any issue. If you have performed these, there is a 99 percent chance that the issue may be resolved.

Errors and their solutions

There are many issues which occur in the production environment and the web administrator has to dig the logs. It's always difficult for the administrator to understand what those exceptions mean and why they are generated in the application. The best way to understand the exception is to go to the logs and check for the first exception which will give you the exact idea of the issue.

Errors can be classified into three major types based on different components of the enterprise application:

- Application
- JVM (memory)
- Database

Let's discuss some of the exceptions and their solutions that help in making the environment stable.

JVM (memory) issues

Nowadays, applications are very resource-intensive; Tomcat instances run out of memory due to these issues. Administrators have to work very hard to fine tune the Tomcat 8 instances and make the environment very stable to run critical web applications on the Internet.

Out of Memory exception

In an enterprise environment, out of memory issues are encountered on a regular basis due to a high memory requirement of applications and the administrator has to tune the JVM. Failure of this causes an Out of Memory exception for the Tomcat instance.

Exception:

```
SEVERE: Servlet.service() for servlet jsp threw exception
java.lang.OutOfMemoryError: Java heap space
```

Reason:

This error may often occur while running an application, which requires high memory-intensive resources. Hence, it causes the Out of Memory exception on the server and leads to an interruption of services.

Solution:

You have to increase the maximum heap size for the Tomcat system. It's important to note that you can only allocate 70 percent of physical memory as JVM memory and 30 percent is reserved for the OS. Check the JVM configuration using the command `jmap` and then increase it in the configuration.

You have to add the following Java parameters in the startup script of Tomcat, which can be found in `TOMCAT_HOME/bin`, to increase the JVM allocation based on the memory requirement and recycle the Tomcat instance.

```
JAVA_OPTS="-Xms512m Xmx1048m
```

OutOfMemoryError: PermGen space

Tomcat administrators often face the problem with the application's permanent object generation, as every application has different requirements of object generation. Hence, application slowness also results in the generation of `OutOfMemoryError: PermGen space` exception in `catalina.out`.

Exception:

```
MemoryError: PermGen space
java.lang.OutOfMemoryError: PermGen space
```

Reason:

The permanent generation is unique because it holds metadata describing user classes. Applications with a large code base can quickly fill up this segment of the heap which causes `java.lang.OutOfMemoryError: PermGen`, no matter how high your `-Xmx` and how much memory you have on the machine.

Note

Java code methods and classes are stored in the permanent generation.

Solution:

The following parameter should be added to the startup script of Tomcat 7. The parameter will increase the permanent generation space at the time of startup of Tomcat 8.

```
-XX:MaxMetaspaceSize=(MemoryValue)m
```

For example:

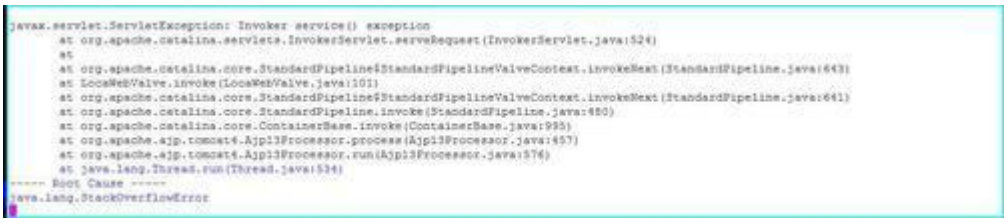
```
-XX:MaxMetaspaceSize=128m
```

Stack over flow exception

We have come across this issue in many applications. This exception is mainly caused due to recursive class loading (improper coding). This issue also causes performance degradation for the application: we observe that the application was working fine an hour ago but then it becomes unresponsive. This is a key indication of the stack overflow exception. The following screenshot shows the error in the logs:

Exception:

```
at java.lang.Thread.run(Thread.java:534)
----- Root Cause -----
java.lang.StackOverflowError
```



Reason:

The exception thrown when the execution stack overflows is because it contains too many nested method calls.

Solution:

You have to increase the value of the `-xss` parameter in the startup file of Tomcat.

```
-Xss=(memory value in k)
```

For example:

```
-Xss=128k
```

By default, the stack overflow exception comes with the value of 64 k followed by the recycle.

Database-related issues

Until now we have discussed various JVM-level issues. Now it's time to discuss common database-related issues.

Broken pipe exception

The broken pipe exception is one of the most common issues reported in the production environment. What does this exception mean? It means that the database connectivity from the J2EE container is terminated. Possible causes for this issue are frequent network disconnects, Ethernet failure on the database, or the J2EE server-level container.

The following screenshot shows the error in the logs:

Exception:

```
at java.lang.Thread.run(Thread.java:619)
Caused by: ClientAbortException: java.net.SocketException: Broken pipe
```

Reason:

This issue is caused due to a connectivity loss between Tomcat 8 and the database.

Solution:

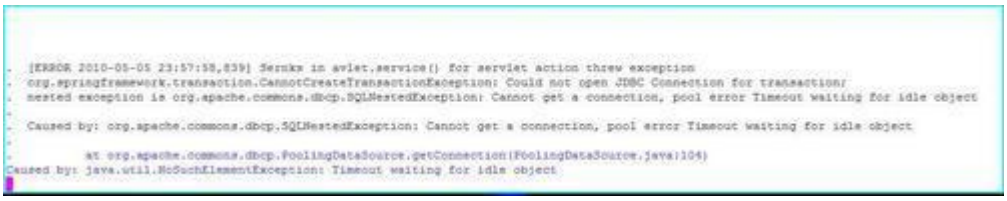
Recycle the Tomcat instance to restore the connectivity.

Timeout waiting for an idle object

Many times, when we click on the application for any transaction, the application displays a blank page after a while. It seems that the application server does not respond but the truth may differ. In many cases, the actual culprit is the database. What happens is the application server sends the request to the database and waits for the response, but the connection is abnormally terminated at the server causing a connection timeout exception. The following screenshot shows the errors in the logs:

Exception:

```
at org.apache.commons.dbcp.PoolingDataSource.getConnection
(PoolingDataSource.java:104)
Caused by: java.util.NoSuchElementException: Timeout waiting for idle object
```



Reason:

This issue is caused due to connection pooling for Tomcat.

Solution:

Change the connection idle values for Tomcat, these settings are in `server.xml`, followed by the recycle.

Database connectivity exception

This kind of issue is often reported in an enterprise environment, where installation of a new application is in process or the migration of an application is in process. It's an issue with an incorrect configuration with JNDI in Tomcat 8.

Exception:

```
java.lang.RuntimeException: Error initializing application. Error Unable to load any
specified brand or the default brand: net.project.persistence.PersistenceException:
Unable to load brand from database.
```

The previous error often indicates that the database could not be accessed.

```
Please check your database configuration or contact your system administrator:
java.sql.SQLException: Error looking up data source for name: jdbc/abc
```

Reason:

Tomcat 8 is unable to connect to the database due to an incorrect JNDI name or the JNDI name doesn't exist.

Solution:

Work with the database administrator to get the correct JNDI name and configure it correctly, followed by a recycle.

Web server benchmarking

Now we know how to troubleshoot problems and find potential solutions in the systems. There is one more point left to discuss, **Web server benchmarking**. Without discussing this topic, troubleshooting in Tomcat 8 cannot be marked as complete. It's a process through which we gauge the performance of a web server, also known as **Load testing**. In this process, we run the server virtually on a heavy load and estimate the real-time performance. This process is very useful if we want to do capacity planning for the web server. There are many tools available for performing load testing on the server such as[**ApacheBench** (**ab**), **JMeter, LoadRunner, OpenSTA**, and so on. Let's discuss the commonly used open source tools such as ApacheBench and JMeter. If we do the benchmarking of the server before the go live stage, then we will face less issues in production support. Also, it helps in improving the performance and designing the scalable environment architecture.

ApacheBench

ApacheBench is a command-line tool for web server benchmarking. It comes under the Apache HTTP server and is very useful when we want to generate only HTTP threads. It's a single thread process.

JMeter

JMeter is one of the widely used open source tools used for load testing. This tool is developed under the Apache Jakarta project. It is capable of generating traffic for JDBC, web services, HTTP, HTTPS, and JMS services. It's a desktop software, which does not support all features of browsers. Following are the advantages of JMeter:

- Portable (can be run on any platform)
- Supports multitasking that allows the administrator to test multiple processes

Note

For more information on JMeter and ApacheBench, visit http://jmeter.apache.org/ and http://httpd.apache.org/docs/2.0/programs/ab.html respectively.

Summary

In this lab, we have discussed different issues faced by the application and web administrators in a real-time environment, how to avoid these issues in the production environment using different techniques with errors and their solutions, thread dump analysis and tools used for analysis, memory issues, steps for troubleshooting real-time problems, and web server benchmarking.

By the end of this lab, the reader will be quite confident to work on the issues that are faced in a real-time environment and I am confident that, by now, they would have resolved some of the major issues in their environments. If not, then they are planning to fix the issues. In the next lab, we will discuss different ways of managing and monitoring Tomcat 8.