Lab 8. Monitoring and Management of Tomcat 8

Monitoring plays a vital role in an IT administrator's life. It makes the life of a web/infrastructure engineer predictable. When I started my career in web infrastructure support, I always wondered, how does my boss know that a process is 90 percent utilized for a particular system or how does he know that a particular process will die after about 90 minutes from now, without logging into the application? Later, I found out that they have set up a monitoring system using various third-party tools available in the market for servers and application monitoring.

In this lab, we will discuss:

- How to monitor Tomcat 8
- Management of applications using the Tomcat Manager
- A third-party utility used for monitoring Tomcat 8

Before we learn how to monitor Tomcat 8, let us understand why monitoring is actually required for any system, as we have configured the systems well for the user.

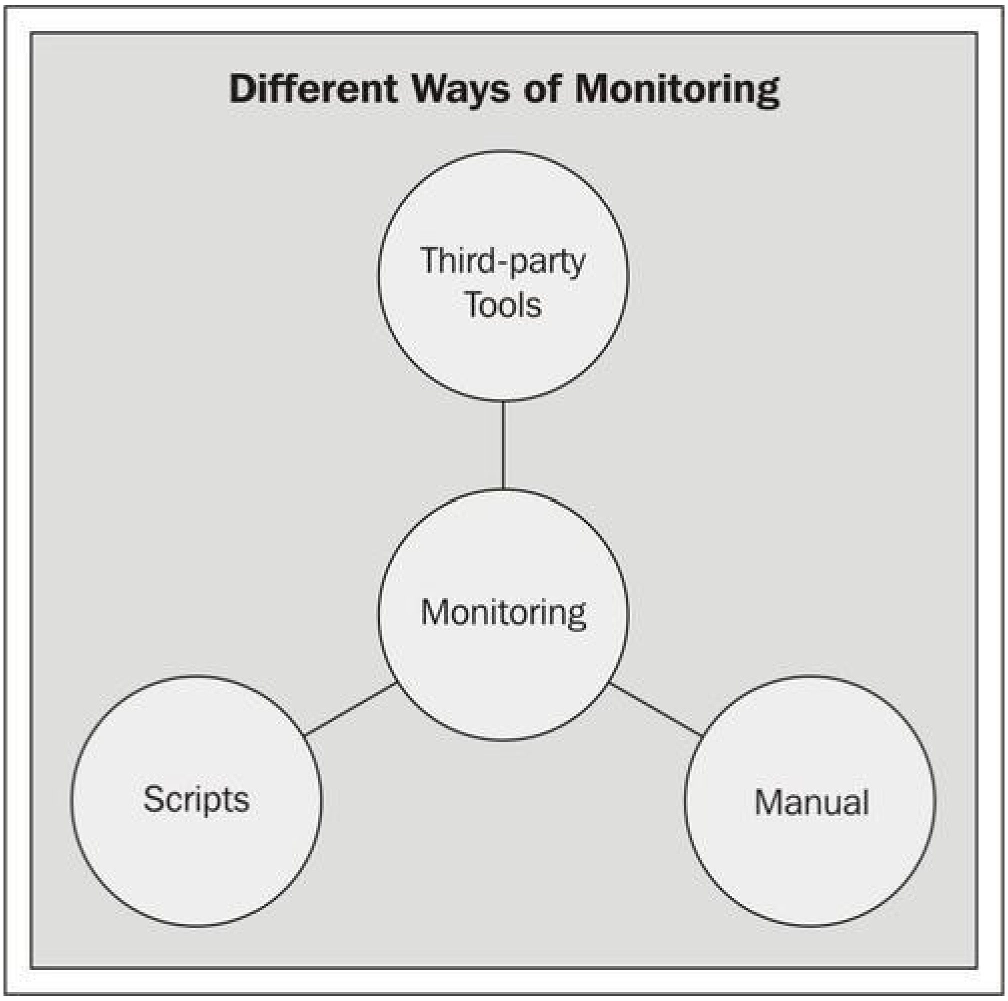
The answer to this question is very tricky. In a real-time environment, the system may break down due to many reasons such as a network glitch, sudden CPU spike, JVM crash, and so on. There are some revenue-generating applications, for example, if bank sites go down, then there will be a huge revenue loss, also administrators will not know unless users start complaining about the issues. This will also have a bad impact on the business. If monitoring systems were set up on the server, the web administrator would get a notification stating that the following systems are going down, and he/she would take the necessary actions to fix the problem. Hence, it minimizes the impact on the application downtime.

Note

IT administrators support thousand of servers, it's practically impossible to validate the system every day. Hence, monitoring is very helpful.

Different ways of monitoring

In today's world, with increasing infrastructure, it becomes very difficult for administrators to manage servers. In order to identify the issue beforehand, and to minimize the downtime, monitors are configured on the system. We can configure multi-level monitoring on the systems, based on the infrastructure requirements for example, the OS, Web, Application, and Database level servers and individual application level. There are different ways of configuring multi-level monitoring. The following figure shows different ways to configure monitoring for any infrastructure:



Monitoring can be mainly done in three ways on a system, which are as follows:

- **Third-party tools**
- Monitoring setups are configured using third-party tools present

```
in the market, such as Wily, SiteScope, Nagios, and so
on.
```

- These kind of monitoring tools are used in an enterprise

```
infrastructure setup, where there are more than 100 servers with
different infrastructure components (domains) such as web,
application, database, filesystem servers, and so on.
```

- **Scripts**
- Scripts are used in monitoring, where a specific use case needs

```
to be monitored, such as the results of how many users are
logged in for a particular interval of time or
application-specific user roles.
```

- Used everywhere in small and big IT organizations.
- **Manual**
- This process is used when any application's performance is slow

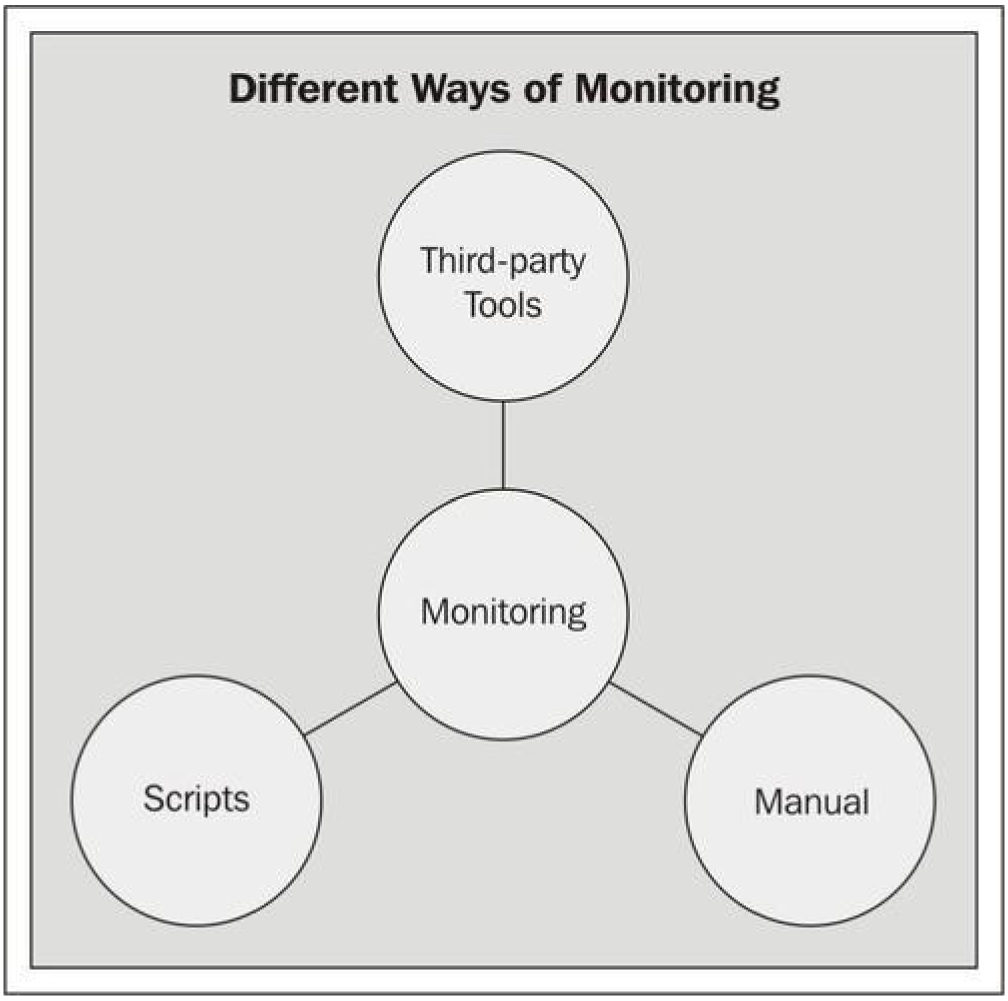
  ```
  for a particular module.
  ```

- Mostly used at the time of troubleshooting and where the number

  ```
  of systems are less than three.
  ```

Different ways of monitoring

In today's world, with increasing infrastructure, it becomes very difficult for administrators to manage servers. In order to identify the issue beforehand, and to minimize the downtime, monitors are configured on the system. We can configure multi-level monitoring on the systems, based on the infrastructure requirements for example, the OS, Web, Application, and Database level servers and individual application level. There are different ways of configuring multi-level monitoring. The following figure shows different ways to configure monitoring for any infrastructure:



Monitoring can be mainly done in three ways on a system, which are as follows:

- **Third-party tools**
- Monitoring setups are configured using third-party tools present

  ```
  in the market, such as Wily, SiteScope, Nagios, and so
  on.
  ```

- These kind of monitoring tools are used in an enterprise

  ```
  infrastructure setup, where there are more than 100 servers with
  different infrastructure components (domains) such as web,
  application, database, filesystem servers, and so on.
  ```

- **Scripts**
- Scripts are used in monitoring, where a specific use case needs

  ```
  to be monitored, such as the results of how many users are
  logged in for a particular interval of time or
  application-specific user roles.
  ```

- Used everywhere in small and big IT organizations.
- **Manual**
- This process is used when any application's performance is slow

  ```
  for a particular module.
  ```

- Mostly used at the time of troubleshooting and where the number

  ```
  of systems are less than three.
  ```

Monitoring setup for a web application and database server

In the previous section, we have discussed the types of monitors, but still we don't know which monitors are configured on these systems. Let's prepare a table for the different infrastructure system monitors and why they are configured. The following table shows the basic monitors, which are normally configured for web application and database servers:

Monitored component	Benefit	Web server	Application server	Database server
CPU	Proactive measure to identify system issues	Yes	Yes	Yes
Physical Memory (RAM)	Proactive measure to identify system issues	Yes	Yes	Yes
JVM	Proactive measure to identify system issues	No	Yes	Yes
HTTP connection	Helps us identify the performance of the web server	Yes	Yes (only if HTTP services are running)	No
AJP connection	Helps us find out the connectivity of the web/application server	Yes	Yes	No
Database connection count	Helps us identify the performance of the database server	No	No	Yes
Connection Idle	Helps us identify the issues of the database server	No	No	Yes
Disk space	Proactive measure to identify system issues (Disk out of space)	Yes	Yes	Yes
Error code in logs	Helps us identify potential issues on systems	No	Yes	Yes

Tomcat Manager in Tomcat 8

The Tomcat Manager is a default tool for managing operations of Apache Tomcat 8. It provides freedom to the IT administrators to remotely manage the application and monitor the systems. Following are the advantages of the Tomcat Manager:

- Allow remote deploy, rollback, start, and stop features for the administrator.
- Provide detailed monitoring status for the application and server.
- Administrators need not stay in the office 24x7. In case of any issues, he/she can log in to the Tomcat Manager to resolve the issue. In short, we can say remote administration of Tomcat becomes very easy for administrators.

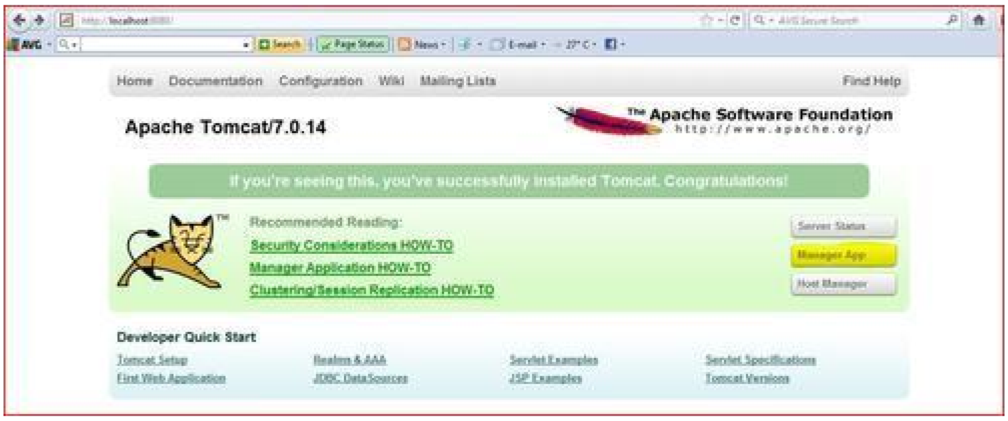
Note

It's not recommended to open the Tomcat Manager from the Internet. In case you have to do so, then we have to enforce strong security policies on Tomcat 8 or we can configure the **Virtual Private Network** (**VPN**) for the administrators.

Perform the following steps to access Tomcat Manager:

1. You can access the Tomcat Manager using the URL http://localhost:8080/. The following screenshot shows the main page for Tomcat 8 with Manager App highlighted.

2. Click on Manager App, it will prompt for the username and password. Provide the credential given in tomcat_user.xml, where tomcat_user.xml can be found in TOMCAT_HOME/CONF.


Home Documentation Configuration Wiki Mailing Lists Find Help

Apache Tomcat/7.0.14

The Apache Software Foundation
http://www.apache.org/

If you're seeing this, you've successfully installed Tomcat. Congratulations!

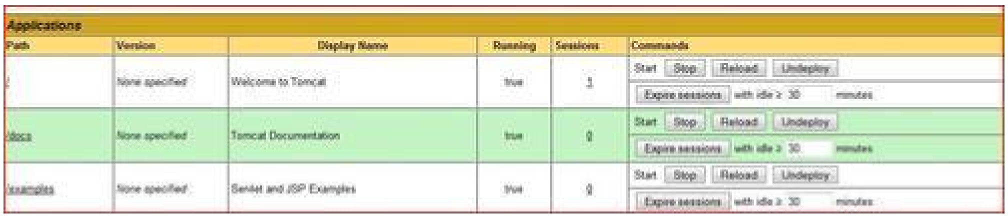
Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Server Status
Manager App
Host Manager

Developer Quick Start

Tomcat Setup	Realms & AAA	Servlet Examples	Servlet Specifications
First Web Application	JDBC DataSources	JSP Examples	Tomcat Versions

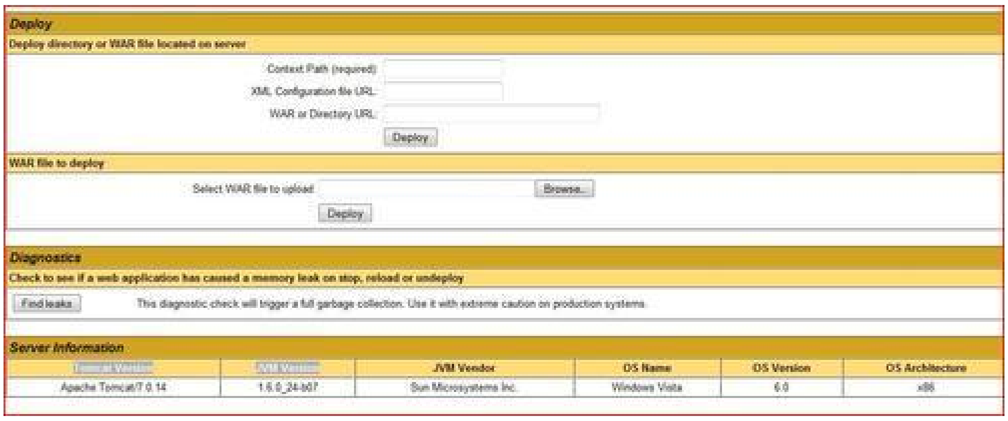
3. The Tomcat Manager console is displayed. If you look at the console, it gives you the complete picture of the application's deployment, server status, diagnostics, server information, and so on. The following screenshot shows the application's status and the different deployment-related tasks performed during application support:


Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

The following screenshot shows the other features of the Tomcat 8 Manager such as:

- Deployment of a new application
- Diagnostic (memory or connection leak)
- Server information


Deploy

Deploy directory or WAR file located on server

Context Path (required):
XML Configuration file URL:
WAR or Directory URL:
Deploy

WAR file to deploy

Select WAR file to upload Browse...
Deploy

Diagnostics

Check to see if a web application has caused a memory leak on stop, reload or undeploy

Find leaks This diagnostic check will trigger a full garbage collection. Use it with extreme caution on production systems.

Server Information

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture
Apache Tomcat/7.0.14	1.6.0_24-b07	Sun Microsystems Inc.	Windows Vista	6.0	x86

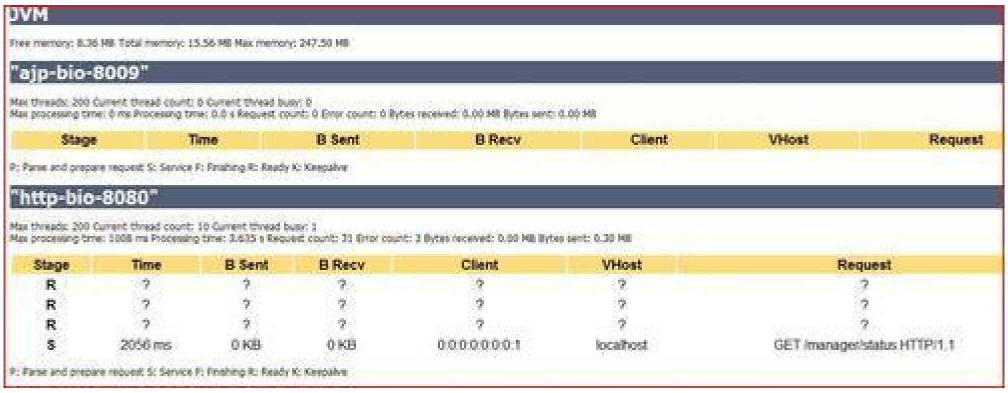
Monitoring in Tomcat 8

Monitoring in Tomcat 8 can be done using the Tomcat Manager. By default, the Tomcat Manager provides the status of the server with a detailed description of requests and their status. This information is very useful to administrators at the time of troubleshooting. Apart from this, the administrator need not log in to the machine for collecting this information. It takes a minimum of 30 minutes to collect the entire information of the application, if you are checking the server status manually, but using the Tomcat Manager, you are getting it online. That's truly amazing and a great help for IT administrators.

Let's discuss the various components used for monitoring that are available in the Tomcat Manager.

Summary of the Server Status of Tomcat 8

The basic synopsis of Tomcat 8 includes the details of the JVM, HTTP, and the HTTPS connection, which can be accessed using the URL `http://localhost:8080/manager/status`. The following screenshot shows the synopsis of Tomcat 8:



JVM

Free memory: 8.36 MB Total memory: 15.56 MB Max memory: 247.50 MB

"ajp-bio-8009"

Max threads: 200 Current thread count: 0 Current thread busy: 0
Max processing time: 0 ms Processing time: 0.0 s Request count: 0 Error count: 0 Bytes received: 0.00 MB Bytes sent: 0.00 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

"http-bio-8080"

Max threads: 200 Current thread count: 10 Current thread busy: 1
Max processing time: 1008 ms Processing time: 3.635 s Request count: 31 Error count: 3 Bytes received: 0.00 MB Bytes sent: 0.30 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
R	?	?	?	?	?	?
R	?	?	?	?	?	?
R	?	?	?	?	?	?
S	2056 ms	0 KB	0 KB	0:0:0:0:0:0:0:1	localhost	GET /manager/status HTTP/1.1

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

The displayed synopsis of Tomcat 8 contains the details of the JVM, HTTP connection, and the AJP connection summary. The details of the output summary for each component are as follows:

- JVM
- **Free memory**
- **Used memory**
- **Total memory**
- Connections on the HTTP port
- **Max threads**
- **Current thread count**
- **Current thread busy**
- **Max processing time** (ms)
- **Processing time** (s)
- **Request count**
- **Error count**
- **Bytes received** (MB)

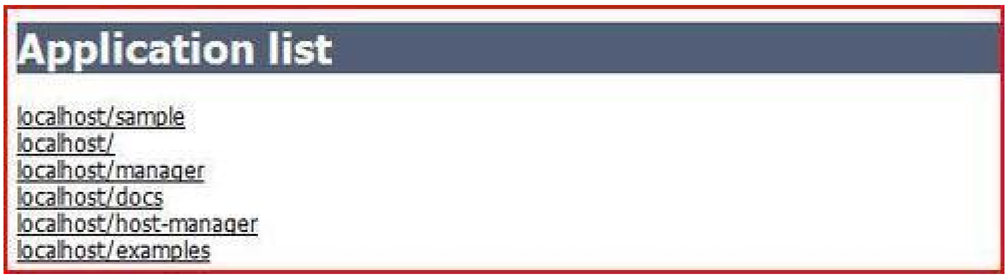
- **Bytes sent** (MB)
- Connections on the AJP
- **Max threads**
- **Current thread count**
- **Current thread busy**
- **Max processing time** (ms)
- **Processing time** (s)
- **Request count**
- **Error count**
- **Bytes received** (MB)
- **Bytes sent** (MB)

Complete Server Status of Tomcat 8

This page gives the complete report of Tomcat 8's status. It includes all the parameters which are part of the **Server Status**. In addition to that, it displays the detailed application list, application response time, servlet response time, and so on. It can be accessed using the URL `http://localhost:8080/manager/status/all`. Let's discuss each component of the dashboard briefly.

Application List

This gives us the list of all applications hosted in Tomcat 8 and their URL mapping for the application's access. The following screenshot shows the application list deployment in the Tomcat 8 instance:

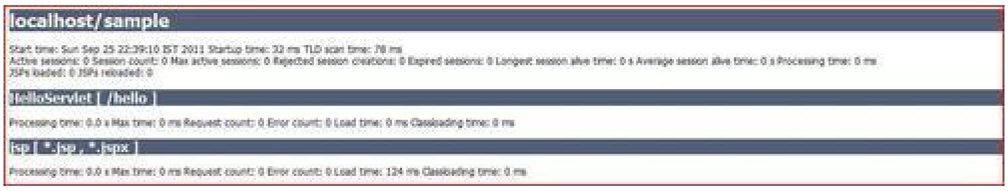


Application list

localhost/sample
localhost/
localhost/manager
localhost/docs
localhost/host-manager
localhost/examples

The following points gives the details of the applications hosted in Tomcat:

- **Application details:** Once we click on the application list, it displays the complete summary of the application including its internal deployment component. The following screenshot shows the internal components with their statuses such as the status of the application response, servlet response, and the JSP responses:



```
localhost/sample
Start time: Sun Sep 25 22:39:10 IST 2011 Startup time: 32 ms TLD scan time: 78 ms
Active sessions: 0 Session count: 0 Max active sessions: 0 Rejected session creations: 0 Expired sessions: 0 Longest session alive time: 0 s Average session alive time: 0 s Processing time: 0 ms
JSPs loaded: 0 JSPs reloaded: 0
HelloServlet [ /hello ]
Processing time: 0.0 s Max time: 0 ms Request count: 0 Error count: 0 Load time: 0 ms Classloading time: 0 ms
jsp [ *.jsp , *.jspx ]
Processing time: 0.0 s Max time: 0 ms Request count: 0 Error count: 0 Load time: 124 ms Classloading time: 0 ms
```

- **Application response:** Application response for a deployed application gives us the current state of the application. For example, the previous screenshot shows the following parameters with reference to the current behavior of the sample application:
- **Start time**
- **Startup time** (ms)
- **TLD scan time** (ms)
- **Active sessions**
- **Session count**
- **Max active sessions**
- **Rejected session creations**
- **Expired sessions**
- **Longest session alive time** (s)
- **Average session alive time** (s)
- **Processing time** (ms)
- **JSPs loaded**
- **JSPs reloaded**
- **Servlet details:** In this section, the dashboard displays the response time of the servlet deployed for a sample application with the following parameters:
- **Processing time** (s)
- **Max time** (ms)
- **Request count**
- **Error count**
- **Load time** (ms)
- **Classloading time** (ms)
- **JSP:** In this section, the dashboard displays the response time of the JSP deployed for a sample application with the following parameters:
- **Processing time** (s)
- **Max time** (ms)
- **Request count**

- **Error count**
- **Load time** (ms)
- **Classloading time** (ms)

JVM

In this section, the dashboard displays the JVM memory utilization for the Tomcat instance. The first column in the following screenshot shows the JVM memory utilization for the sample application with the following parameters:

- **Free memory**
- **Used memory**
- **Total memory**

Connections on the HTTP port (8080)

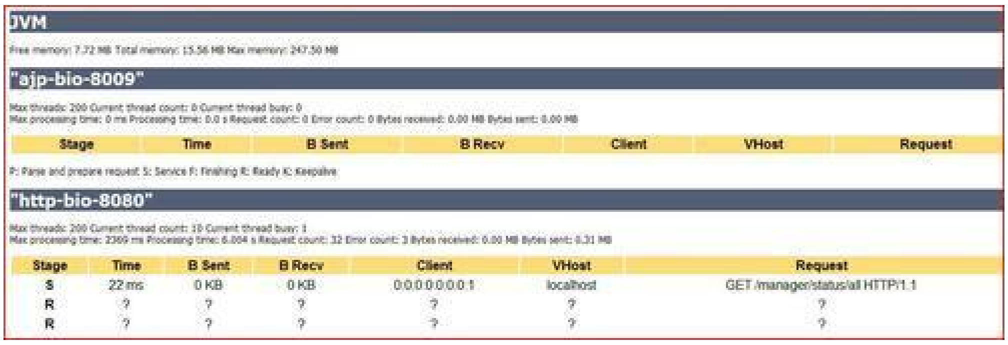
In this section, the dashboard displays the HTTP connection status for the Tomcat instance. The second column in the following screenshot shows the HTTP connection status for the application sample with the following parameters:

- **Max threads**
- **Current thread count**
- **Current thread busy**
- **Max processing time** (ms)
- **Processing time** (s)
- **Request count**
- **Error count**
- **Bytes received** (MB)
- **Bytes sent** (MB)

Connections on the AJP

In this section, the dashboard displays the AJP connection status for the Tomcat instance. The third column in the following screenshot shows the status of AJP connection for the application sample with the following parameters:

- **Max threads**
- **Current thread count**
- **Current thread busy**
- **Max processing time** (ms)
- **Processing time** (s)
- **Request count**
- **Error count**
- **Bytes received** (MB)
- **Bytes sent** (MB)



JVM

Free memory: 7.72 MB Total memory: 15.56 MB Max memory: 247.50 MB

"ajp-bio-8009"

Max threads: 200 Current thread count: 0 Current thread busy: 0
Max processing time: 0 ms Processing time: 0.0 s Request count: 0 Error count: 0 Bytes received: 0.00 MB Bytes sent: 0.00 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

"http-bio-8080"

Max threads: 200 Current thread count: 10 Current thread busy: 1
Max processing time: 2369 ms Processing time: 6.004 s Request count: 32 Error count: 3 Bytes received: 0.00 MB Bytes sent: 0.31 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
S	22 ms	0 KB	0 KB	0:0:0:0:0:0:0:1	localhost	GET /manager/status/all HTTP/1.1
R	?	?	?	?	?	?
R	?	?	?	?	?	?

JConsole configuration on Tomcat 8

JConsole is one of the best monitoring utilities that comes with JDK 1.5 or later. The full form of the JConsole is the[**Java Monitoring and Management Console**. It's a graphical tool, which gives complete details of the application and server performance. It gives us the following information about the application hosted in Tomcat 8:

- Detect low memory
- Enable or disable the GC and class loading verbose tracing
- Detect deadlocks
- Control the log level of any loggers in an application
- Access the OS resources---Sun's platform extension
- Manage an application's **Managed Beans** (**MBeans**)

Remote JMX enabling

In order to use the JConsole for Tomcat 8 monitoring, we have to enable the **Java Management Extension** (**JMX**) on Tomcat 8. By doing this, we can monitor the Tomcat 8 server details from our desktop machine also, or in simple terms, we can monitor the server status remotely without logging into the server machine. It gives great flexibility to the administrator to work from any location and troubleshoot the problem. In order to enable it in Tomcat 8, we have to add the `CATALINA_OPTS` parameter in `catalina.sh`. By default, the following values are added to enable the details:

```
CATALINA_OPTS=-Dcom.sun.management.jmxremote \ -
Dcom.sun.management.jmxremote.port=%my.jmx.port% \ -
Dcom.sun.management.jmxremote.ssl=false \ -
Dcom.sun.management.jmxremote.authenticate=false
```

Let's do the real-time configuration on Tomcat 8 and understand the meaning of each parameter:

```
CATALINA_OPTS="-Djava.awt.headless=true -Xmx128M -server -
Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote. port=8086 -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false"
```

- `-Djava.awt.headless:` It is a system configuration option that helps the graphics rendering program to accept the graphics console and redirects the program to work in the command-line mode. It is very useful while connecting to the remote server.
- `-Dcom.sun.management.jmxremote:` This JMX allows the host to connect to the system.
- `-Dcom.sun.management.jmxremote.port:` It defines the port where your **Remote Method Invocation** (**RMI**) is connected.
- `-Dcom.sun.management.jmxremote.authenticate:` It defines the authentication mechanism for the connection.
- `-Dcom.sun.management.jmxremote.ssl:` It defines the protocol used for communication. If it is set to `false`, then, by default, it uses the HTTP protocol.

How to connect to the JConsole

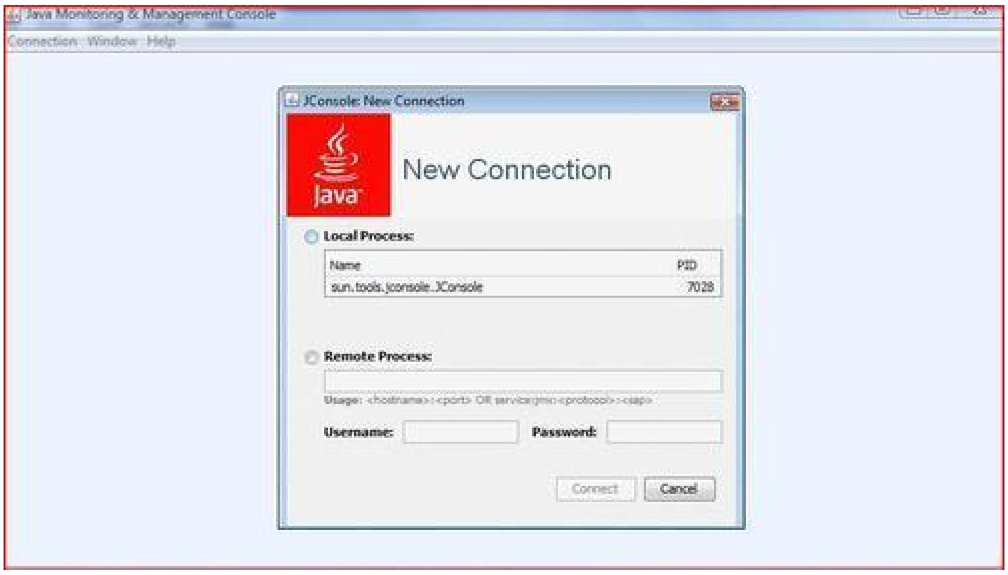
Once Tomcat 8 configurations are done, it's time to connect to Tomcat 8 through the JConsole remotely using the command `jconsole`, as follows. It will open the GUI interface. We have to provide the IP address and port for the server where we want to connect; in our case, it's `localhost` and `8086`. The following screenshot shows the default console for the JConsole:

```
[root@localhost bin] # jconsole
```

Note

`jconsole` can be found in `JAVA_HOME/bin`.

If we have included `JAVA_HOME/bin` in the path, then we can execute this command from anywhere in the system.



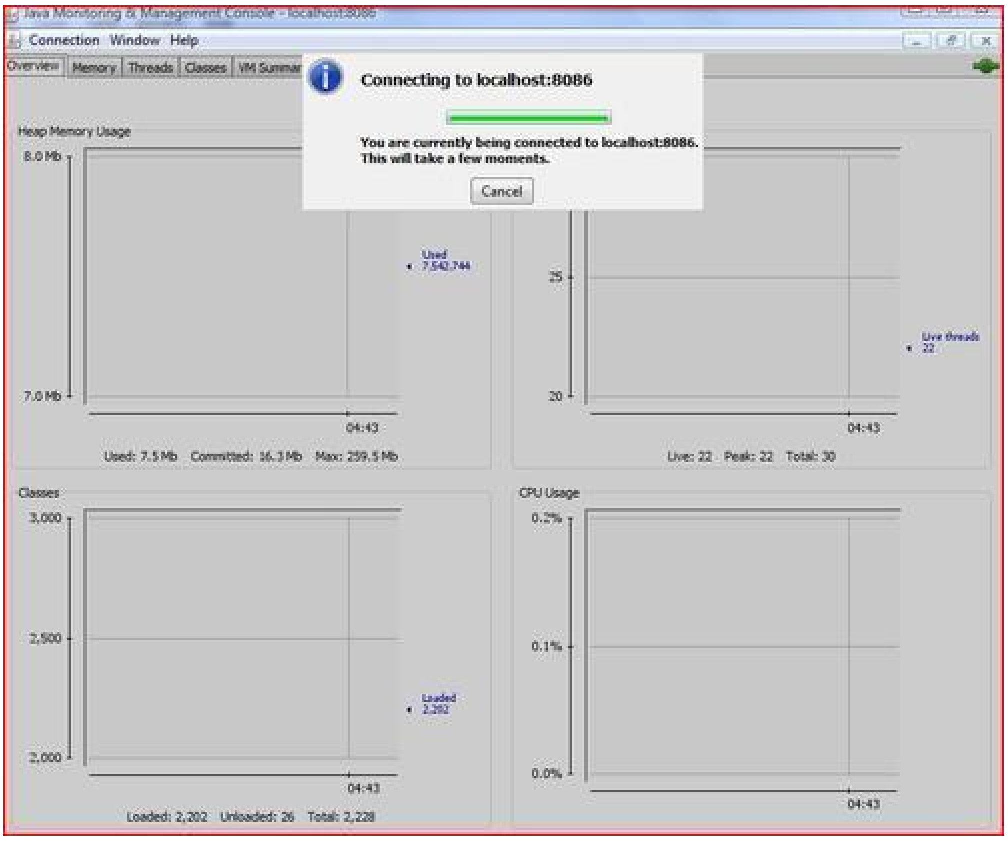
Name	PID
sun.tools.jconsole.JConsole	7028

Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

The following screenshot shows the JConsole connecting to Tomcat 8 using the port 8086:

Note

To connect to remote servers, we have to enable the firewall to allow the JConsole port on the server.


Connecting to localhost:8086

You are currently being connected to localhost:8086.
This will take a few moments.

Java Monitoring & Management Console - localhost:8086
Connection Window Help
Overview Memory Threads Classes VM Summary
Cancel
Heap Memory Usage
8.0 Mb
7.0 Mb
Used 7,542,744
04:43
Used: 7.5 Mb Committed: 16.3 Mb Max: 259.5 Mb
25
20
Live threads 22
Live: 22 Peak: 22 Total: 30
Classes
3,000
2,500
2,000
Loaded 2,202
Loaded: 2,202 Unloaded: 26 Total: 2,228
CPU Usage
0.2%
0.1%
0.0%

Once the system is connected, it gives the complete overview of the system such as the CPU, memory, thread, and classes. The following screenshot shows the details. We can also do a deep analysis for the following components of Java-based applications:

- Memory
- Thread
- Classes
- MBeans

The advantages of using this tool are:

- Online analysis of the application
- Customized report for the analysis
- Deadlock can be retrieved in the systems

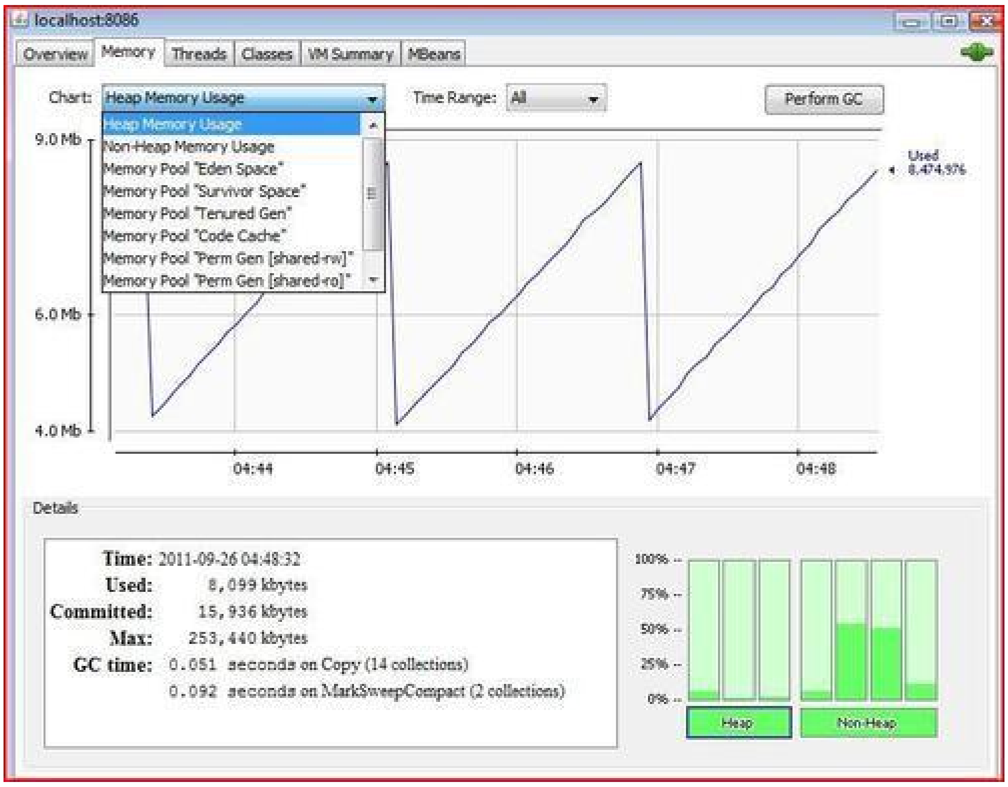
Different tabs for the JConsole and their features

We will now discuss the different components of monitoring for the JConsole.

Memory overview

It's necessary for web administrators to analyze the memory status for Tomcat 8 to avoid future issues with the server. The following screenshot shows the real-time heap memory utilization for Tomcat 8. This tab provides the following features:

- Graphical presentation of memory with their JVM footprints
- Customization of the Memory chart based on the requirement analysis
- Ability to perform the GC

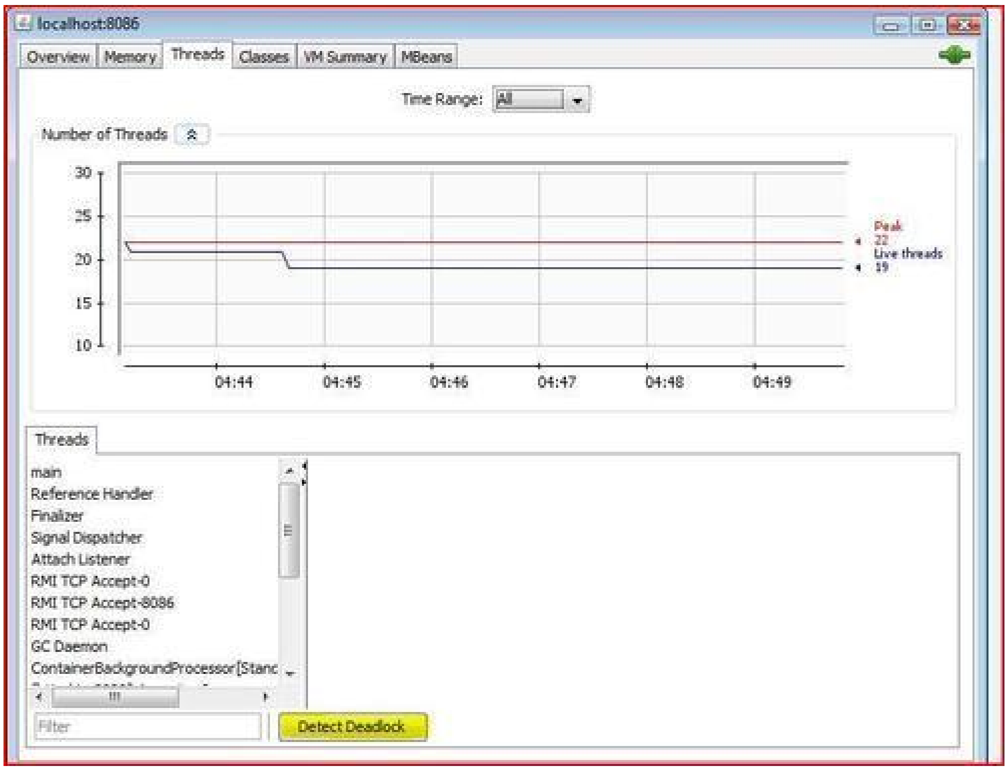


```
     Time: 2011-09-26 04:48:32
     Used:     8,099 kbytes
Committed:    15,936 kbytes
      Max:   253,440 kbytes
  GC time: 0.051 seconds on Copy (14 collections)
           0.092 seconds on MarkSweepCompact (2 collections)
```

Threads overview

This tab gives the complete picture of the server threads and the web application hosted in Tomcat 8, for a particular instance. The following screenshot shows the live status of the threads utilization, and the[**Deadlock Detection** button is highlighted. In real-time, this thread analysis tool is a very handy tool for the administrators. Following are the features offered by the **Threads** tab:

- Graphical presentation of threads and their picture
- Individual thread analysis with their status
- Deadlock detection

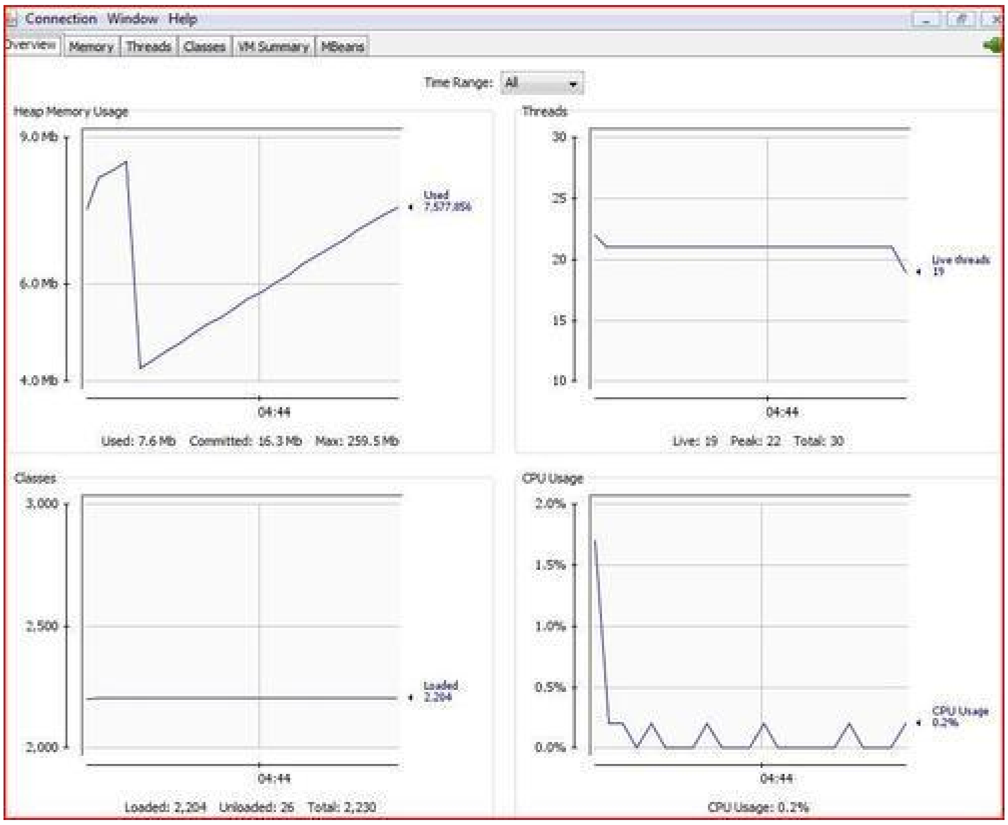


main
Reference Handler
Finalizer
Signal Dispatcher
Attach Listener
RMI TCP Accept-0
RMI TCP Accept-8086
RMI TCP Accept-0
GC Daemon
ContainerBackgroundProcessor[Stanc

VM Summary and Overview

These two tabs are very important for an administrator. In practice, it's not possible for the administrator to view each and every component of the application every time. What the administrator does is, he/she checks for the overall performance of the system. If any anomalies are found, they will drill down the component. Following are the features of these tabs:

- Complete summary of the instances (**Heap Memory Usage, Threads, CPU Usage, Classes**)
- VM argument summary



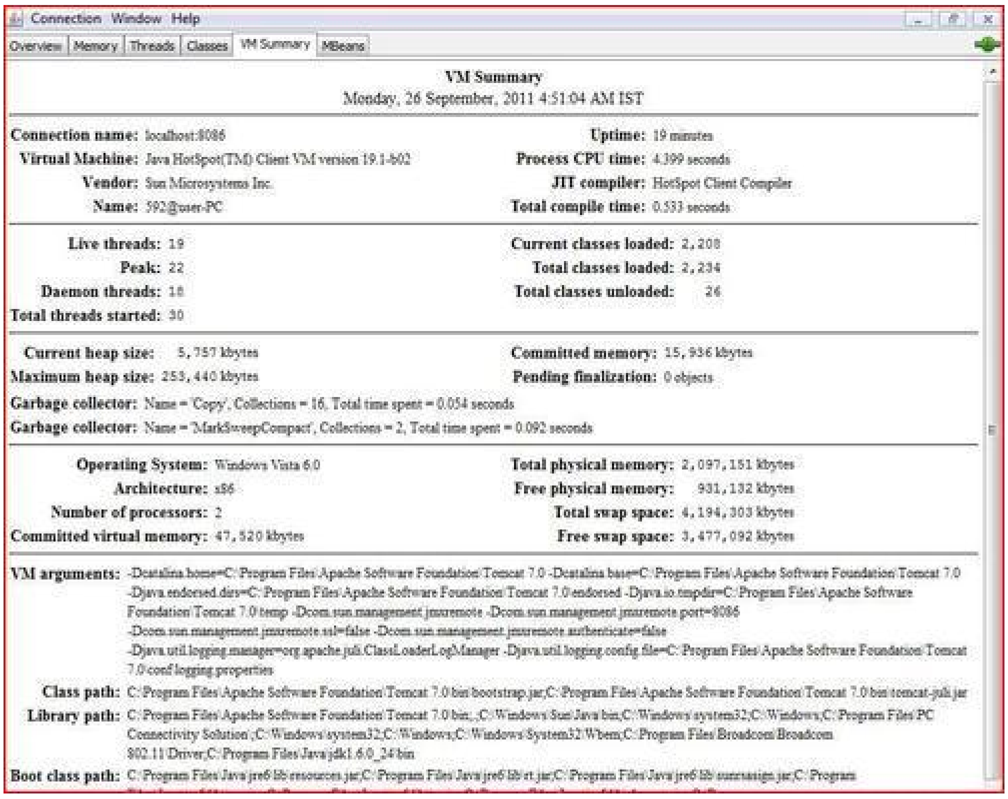
Used: 7.6 Mb Committed: 16.3 Mb Max: 259.5 Mb

Live: 19 Peak: 22 Total: 30

Loaded: 2,204 Unloaded: 26 Total: 2,230

CPU Usage: 0.2%

The previous screenshot shows the real-time status of Tomcat 8. It consists of four graphs displaying the real-time utilization of the heap, threads, classes and CPU usage for the Tomcat 8 instance. On the other hand, the following screenshot shows the complete summary of the Tomcat instance:

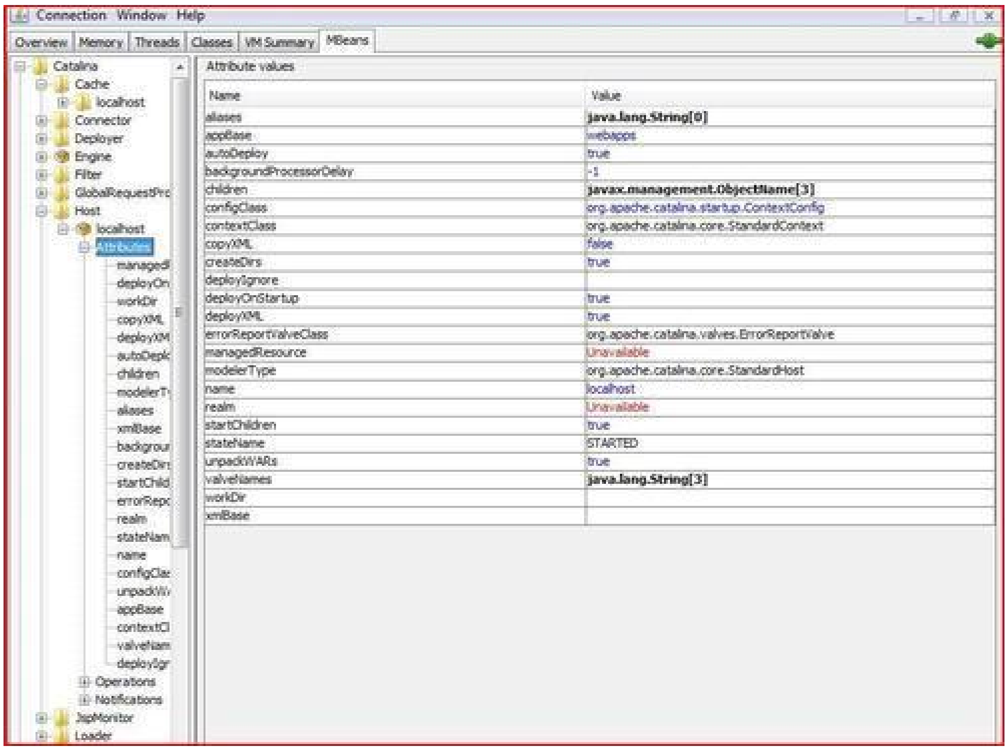


```
VM Summary
Monday, 26 September, 2011 4:51:04 AM IST

Connection name: localhost:8086                              Uptime: 19 minutes
Virtual Machine: Java HotSpot(TM) Client VM version 19.1-b02 Process CPU time: 4.399 seconds
         Vendor: Sun Microsystems Inc.                       JIT compiler: HotSpot Client Compiler
           Name: 592@user-PC                                 Total compile time: 0.533 seconds

         Live threads: 19                Current classes loaded: 2,208
                 Peak: 22                  Total classes loaded: 2,234
      Daemon threads: 18                Total classes unloaded:    26
Total threads started: 30

  Current heap size:   5,757 kbytes      Committed memory: 15,936 kbytes
Maximum heap size: 253,440 kbytes    Pending finalization: 0 objects
Garbage collector: Name = 'Copy', Collections = 16, Total time spent = 0.054 seconds
Garbage collector: Name = 'MarkSweepCompact', Collections = 2, Total time spent = 0.092 seconds

      Operating System: Windows Vista 6.0          Total physical memory: 2,097,151 kbytes
          Architecture: x86                         Free physical memory:   931,132 kbytes
  Number of processors: 2                             Total swap space: 4,194,303 kbytes
Committed virtual memory: 47,520 kbytes                Free swap space: 3,477,092 kbytes

VM arguments: -Dcatalina.home=C:\Program Files\Apache Software Foundation\Tomcat 7.0 -Dcatalina.base=C:\Program Files\Apache Software Foundation\Tomcat 7.0
              -Djava.endorsed.dirs=C:\Program Files\Apache Software Foundation\Tomcat 7.0\endorsed -Djava.io.tmpdir=C:\Program Files\Apache Software
              Foundation\Tomcat 7.0\temp -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=8086
              -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false
              -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.util.logging.config.file=C:\Program Files\Apache Software Foundation\Tomcat
              7.0\conf\logging.properties
  Class path: C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin\bootstrap.jar;C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin\tomcat-juli.jar
Library path: C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin;.;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;C:\Program Files\PC
              Connectivity Solution\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Program Files\Broadcom\Broadcom
              802.11\Driver;C:\Program Files\Java\jdk1.6.0_24\bin
Boot class path: C:\Program Files\Java\jre6\lib\resources.jar;C:\Program Files\Java\jre6\lib\rt.jar;C:\Program Files\Java\jre6\lib\sunrsasign.jar;C:\Program
```

MBeans

This tab gives you the complete picture of **Managed Beans** (**MBeans**) deployed in the Tomcat instance. It includes both Tomcat and application-level MBeans. The following screenshot shows the attributes of **MBeans**. It's very useful if a particular MBean is the source for an issue. Following are the advantages of the **MBeans** tab:

- All parameters used in one tab
- Easy-to-deploy, rollback, and invoke
- We can create a user at the database level using MBeans
- We can create notifications for events using MBeans
- Configuration for resources can be done dynamically

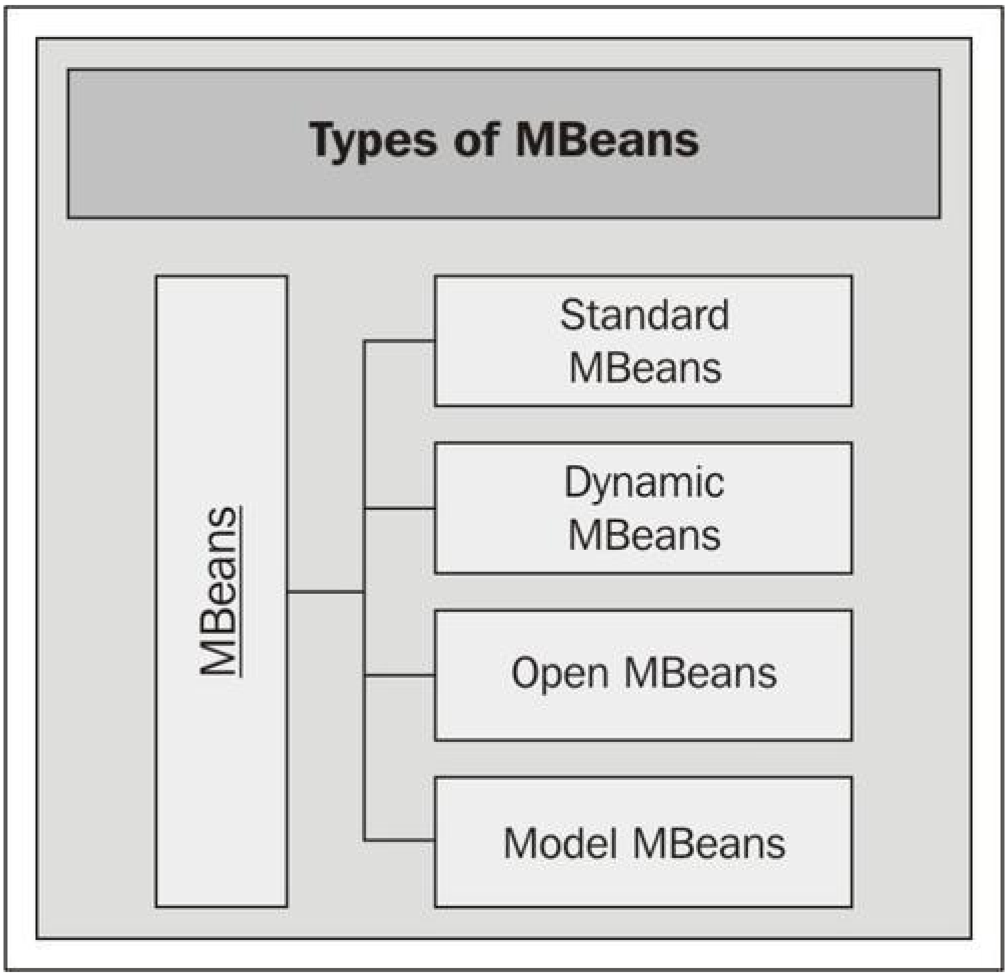


Name	Value
aliases	**java.lang.String[0]**
appBase	webapps
autoDeploy	true
backgroundProcessorDelay	-1
children	**javax.management.ObjectName[3]**
configClass	org.apache.catalina.startup.ContextConfig
contextClass	org.apache.catalina.core.StandardContext
copyXML	false
createDirs	true
deployIgnore	
deployOnStartup	true
deployXML	true
errorReportValveClass	org.apache.catalina.valves.ErrorReportValve
managedResource	Unavailable
modelerType	org.apache.catalina.core.StandardHost
name	localhost
realm	Unavailable
startChildren	true
stateName	STARTED
unpackWARs	true
valveNames	**java.lang.String[3]**
workDir	
xmlBase	

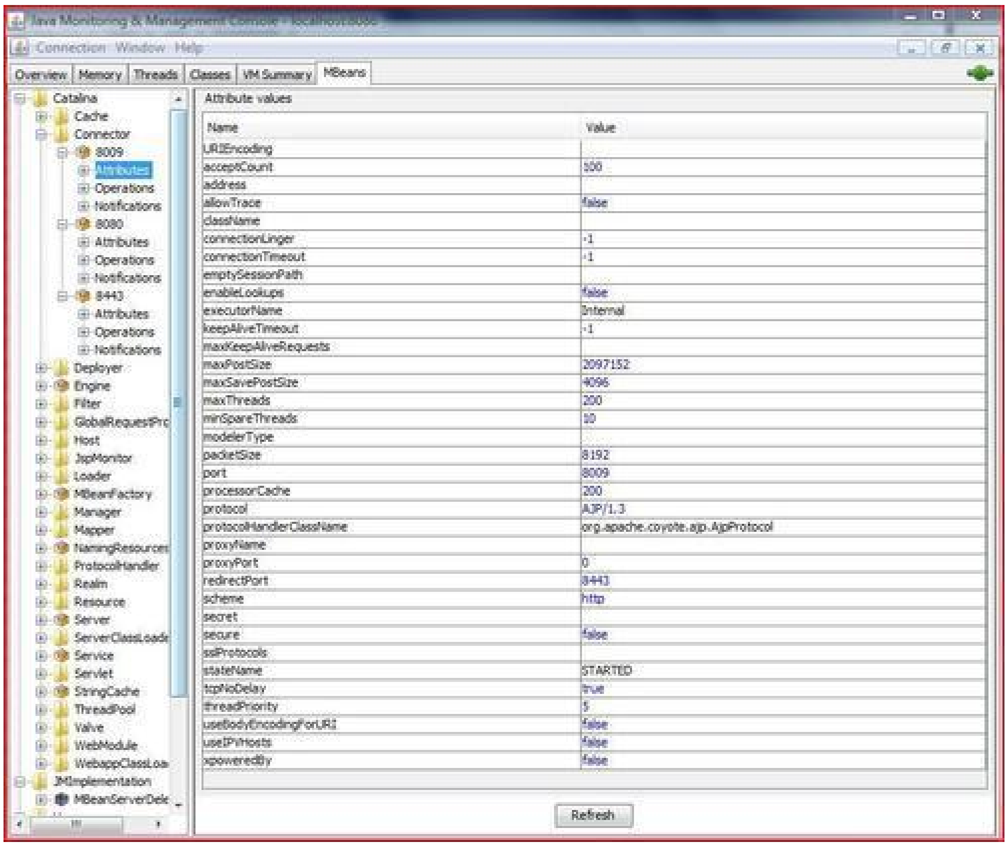
Types of MBeans

There are four types of MBeans. The following figure shows the different types of MBeans. Let's discuss each MBean briefly:

- **Standard MBeans:** A standard MBean is a combination of an MBean interface and a class, where the interface defines the entire list of attributes and operations, while the class provides the functionalities for communication for a remote interface. It is one of the simplest MBeans.
- **Dynamic MBeans:** A dynamic MBean implements a separate interface (a specific method) and can be invoked at runtime.
- **Open MBeans:** It is a composition of Dynamic MBeans and the universal dataset used for manageability.
- **Model MBeans:** It is a composition of Dynamic MBeans with complete access to configurable parameters at runtime and self-described methods. This MBean requires classes.



Let's take an example of the Connectors deployed in Tomcat 8, where we can configure and perform operations on the Connectors remotely using MBeans. By default, the HTTP and AJP Connectors are configured. In our example, we have the HTTPS Connector also configured. The following screenshot shows the three Connectors HTTP, AJP, and HTTPS:

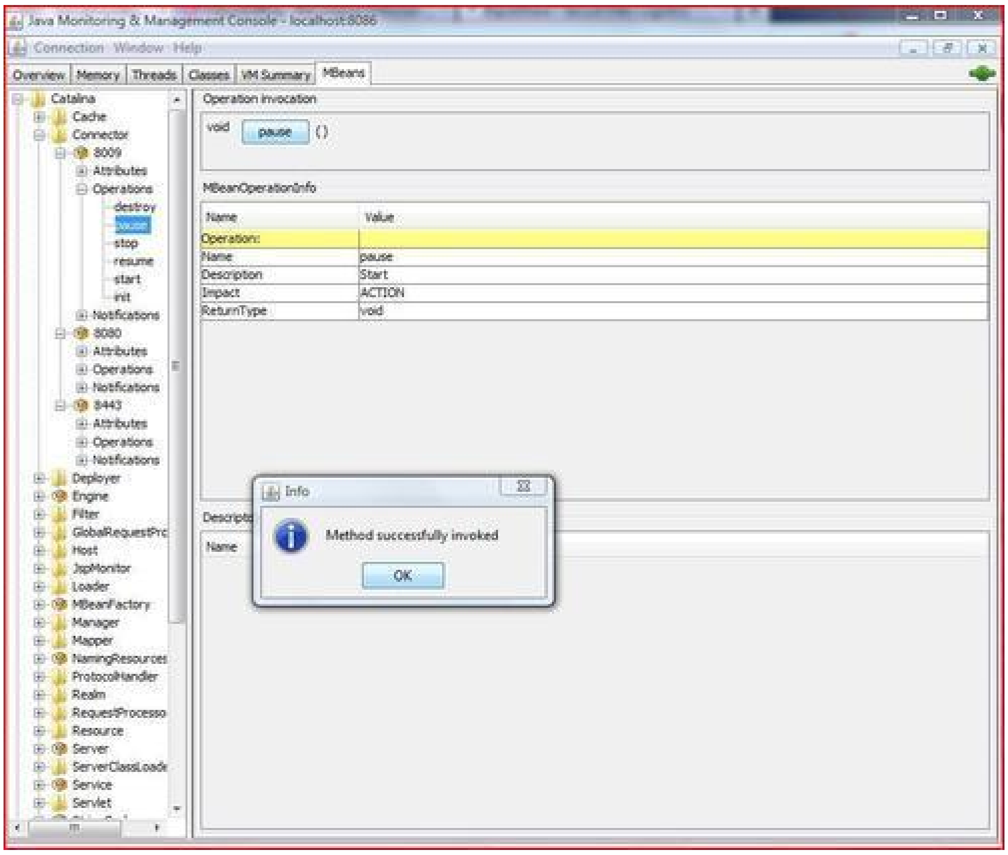


Name	Value
URIEncoding	
acceptCount	100
address	
allowTrace	false
className	
connectionLinger	-1
connectionTimeout	-1
emptySessionPath	
enableLookups	false
executorName	Internal
keepAliveTimeout	-1
maxKeepAliveRequests	
maxPostSize	2097152
maxSavePostSize	4096
maxThreads	200
minSpareThreads	10
modelerType	
packetSize	8192
port	8009
processorCache	200
protocol	AJP/1.3
protocolHandlerClassName	org.apache.coyote.ajp.AjpProtocol
proxyName	
proxyPort	0
redirectPort	8443
scheme	http
secret	
secure	false
sslProtocols	
stateName	STARTED
tcpNoDelay	true
threadPriority	5
useBodyEncodingForURI	false
useIPVHosts	false
xpoweredBy	false

If we observe the **Connector** folder, we can view the three sublevels for each Connector. Following are the functions of each section:

- **Attributes:** This section contains information about the different parameters loaded in the memory during the startup of the Tomcat instance. In short, we can say that information of the configured parameter is loaded at runtime.
- **Operations:** The functionality of this mode is to perform runtime operations for MBeans. Following are the different operations we can perform:
- Destroy
- Start
- Stop
- Init
- Resume
- Pause

```
The following screenshot shows the pause operation successfully
invoked for the HTTP Connector for Tomcat 8:
```



Name	Value
Operation:	
Name	pause
Description	Start
Impact	ACTION
ReturnType	void

Method successfully invoked

- **Nofitications:** It is used to configure notifications for an event such as the state of MBeans, deadlock, and so on. To enable notifications, we have to subscribe to it.

Note

For more information on monitoring, please refer to the link http://java.sun.com/developer/technicalArticles/J2SE/monitoring/.

Summary

In this lab, we have discussed the various processes of monitoring in Tomcat 8 and their components using the Tomcat Manager and JConsole, such as different ways of monitoring, how monitoring is done in Tomcat 7, JConsole, and how it is used. In the next lab, we will discuss the high availability setup for Tomcat 8 using clustering, load balancing, high availability concepts, architecture design, scalability, and so on.