# Lab 6. Logging in Tomcat 8


Logging services play a vital role in the life of the administrator and developer to manage the application from the phase of development to production issues. It's the logging services that help you to find the actual problem in the web application. Also, it plays an essential role in performance tuning for many applications.

In this lab, we will discuss:

- Logging services in Tomcat 8
- JULI
- Log4j
- Log level
- Valve component
- Analysis of logs

## JULI

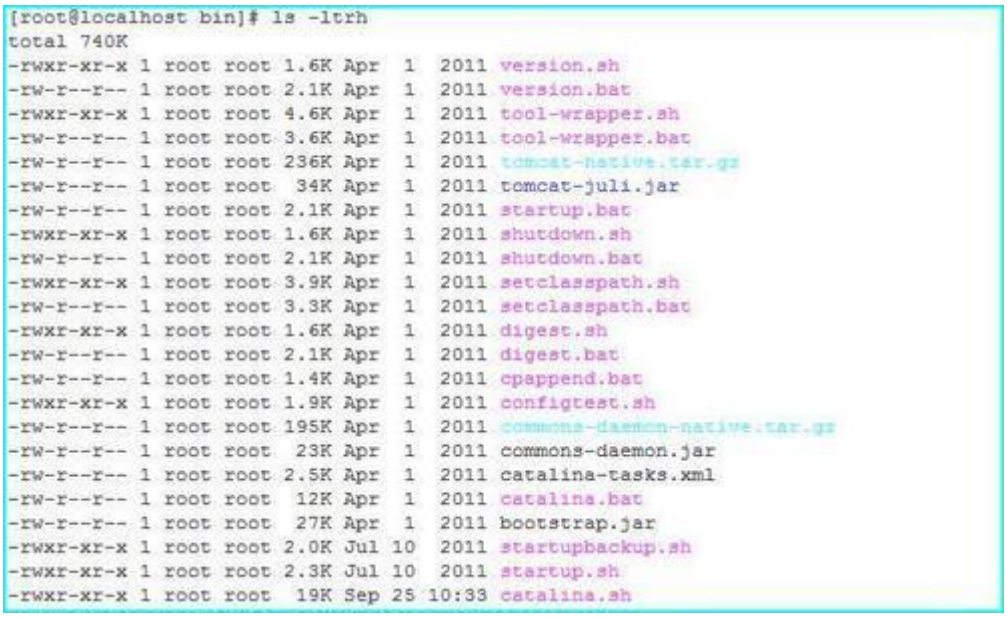
Previous versions of Tomcat (until 5.x) use Apache common logging services for generating logs. A major disadvantage with this logging mechanism is that it can handle only a single JVM configuration and it makes it difficult to configure separate logging for each class loader for independent applications. In order to resolve this issue, Tomcat developers have introduced a separate API for the Tomcat 6 version, that comes with the capability of capturing each class loader activity in the Tomcat logs. It is based on the `java.util.logging` framework.

By default, Tomcat 8 uses its own **Java logging API** to implement logging services. This is also called **JULI**. This API can be found in `TOMCAT_HOME/bin` of the Tomcat 8 directory structures ( `tomcat-juli.jar` ). The following screenshot shows the directory structure of the `bin` directory where `tomcat-juli.jar` is placed. JULI also provides the feature for custom logging for each web application, and it also supports private per-application logging configurations. With the enhanced feature of separate class loader logging, it also helps in detecting memory issues while unloading the classes at runtime.

### Note

For more information on JULI and the class loading issue, please refer to http://tomcat.apache.org/tomcat-7.0-doc/logging.html and http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html respectively.

---




## JULI

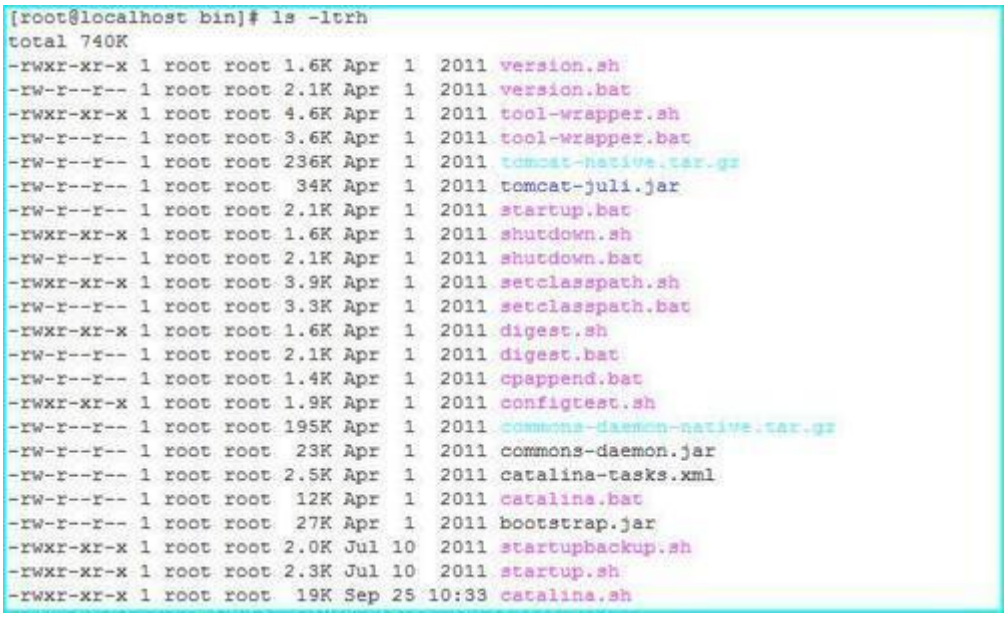
Previous versions of Tomcat (until 5.x) use Apache common logging services for generating logs. A major disadvantage with this logging mechanism is that it can handle only a single JVM configuration and it makes it difficult to configure separate logging for each class loader for independent applications. In order to resolve this issue, Tomcat developers have introduced a separate API for the Tomcat 6 version, that comes with the capability of capturing each class loader activity in the Tomcat logs. It is based on the `java.util.logging` framework.

By default, Tomcat 8 uses its own **Java logging API** to implement logging services. This is also called **JULI**. This API can be found in `TOMCAT_HOME/bin` of the Tomcat 8 directory structures ( `tomcat-juli.jar` ). The following screenshot shows the directory structure of the `bin` directory where `tomcat-juli.jar` is placed. JULI also provides the feature for custom logging for each web application, and it also supports private per-application logging configurations. With the enhanced feature of separate class loader logging, it also helps in detecting memory issues while unloading the classes at runtime.

### Note

For more information on JULI and the class loading issue, please refer to http://tomcat.apache.org/tomcat-7.0-doc/logging.html and http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html respectively.

---



```
[root@localhost bin]# ls -ltrh
total 740K
-rwxr-xr-x 1 root root 1.6K Apr  1  2011 version.sh
-rw-r--r-- 1 root root 2.1K Apr  1  2011 version.bat
-rwxr-xr-x 1 root root 4.6K Apr  1  2011 tool-wrapper.sh
-rw-r--r-- 1 root root 3.6K Apr  1  2011 tool-wrapper.bat
-rw-r--r-- 1 root root 236K Apr  1  2011 tomcat-native.tar.gz
-rw-r--r-- 1 root root  34K Apr  1  2011 tomcat-juli.jar
-rw-r--r-- 1 root root 2.1K Apr  1  2011 startup.bat
-rwxr-xr-x 1 root root 1.6K Apr  1  2011 shutdown.sh
-rw-r--r-- 1 root root 2.1K Apr  1  2011 shutdown.bat
-rwxr-xr-x 1 root root 3.9K Apr  1  2011 setclasspath.sh
-rw-r--r-- 1 root root 3.3K Apr  1  2011 setclasspath.bat
-rwxr-xr-x 1 root root 1.6K Apr  1  2011 digest.sh
-rw-r--r-- 1 root root 2.1K Apr  1  2011 digest.bat
-rw-r--r-- 1 root root 1.4K Apr  1  2011 cpappend.bat
-rwxr-xr-x 1 root root 1.9K Apr  1  2011 configtest.sh
-rw-r--r-- 1 root root 195K Apr  1  2011 commons-daemon-native.tar.gz
-rw-r--r-- 1 root root  23K Apr  1  2011 commons-daemon.jar
-rw-r--r-- 1 root root 2.5K Apr  1  2011 catalina-tasks.xml
-rw-r--r-- 1 root root  12K Apr  1  2011 catalina.bat
-rw-r--r-- 1 root root  27K Apr  1  2011 bootstrap.jar
-rwxr-xr-x 1 root root 2.0K Jul 10  2011 startupbackup.sh
-rwxr-xr-x 1 root root 2.3K Jul 10  2011 startup.sh
-rwxr-xr-x 1 root root  19K Sep 25 10:33 catalina.sh
```

## Loggers, appenders, and layouts

There are some important components of logging which we use at the time of implementing the logging mechanism for applications. Each term has its individual importance in tracking the events of the application. Let's discuss each term individually to find out their usage:

- **Loggers:** It can be defined as the logical name for the log file. This logical name is written in the application code. We can configure an independent logger for each application.
- **Appenders:** The process of generating logs is handled by appenders. There are many types of appenders, such as[ **FileAppender, ConsoleAppender, SocketAppender**, and so on, which are available in log4j. The following are some examples of appenders for log4j:

```
log4j.appender.CATALINA=org.apache.log4j.DailyRollingFileAppender
log4j.appender.CATALINA.File=${catalina.base}/logs/catalina.out
log4j.appender.CATALINA.Append=true
log4j.appender.CATALINA.Encoding=UTF-8
```

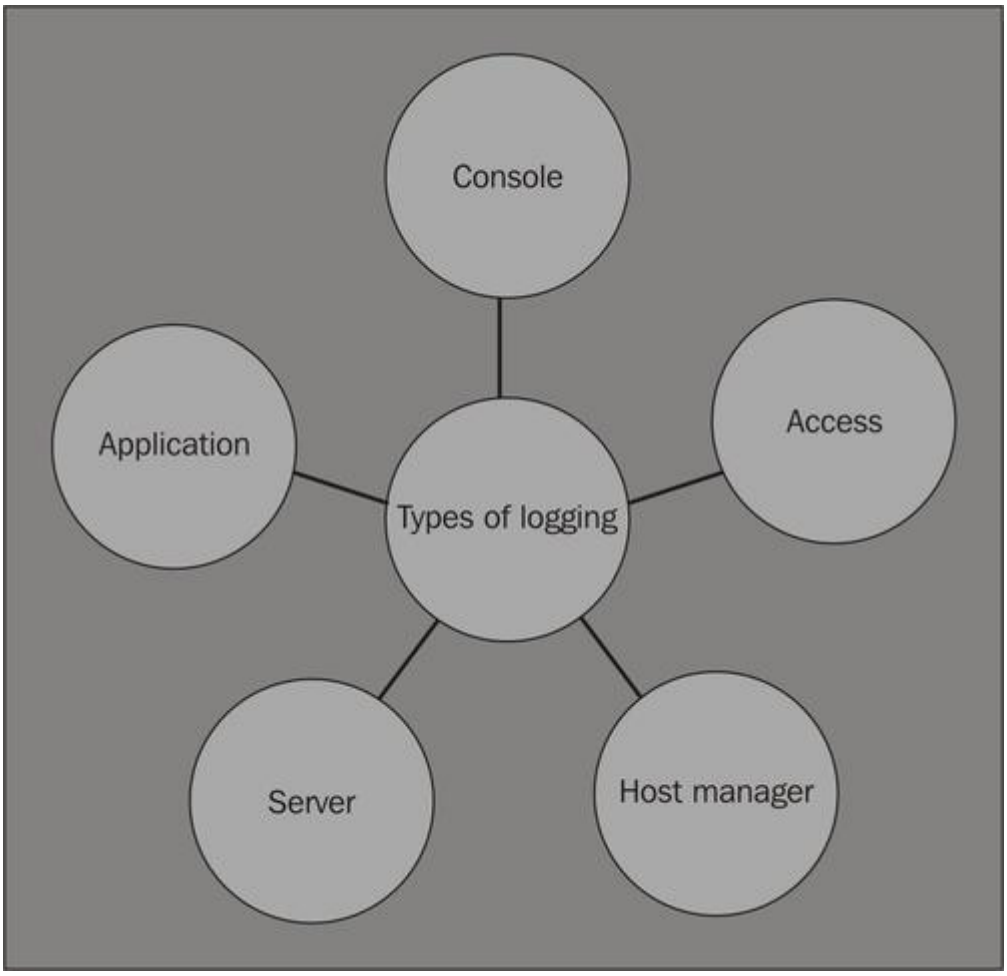
```
The previous four lines of appender define the
DailyRollingFileAppender in log4j, where the filename is
`catalina.out`. These logs will have UTF-8 encoding
enabled.
```

**Note**

```
If` log4j.appender.CATALINA.append=false`, then logs will
not get updated in the log files.


```
# Roll-over the log once per day
```
```

```
log4j.appender.CATALINA.DatePattern='.'dd-MM-yyyy'.log'
log4j.appender.CATALINA.layout = org.apache.log4j.PatternLayout
log4j.appender.CATALINA.layout.ConversionPattern = %d [%t] %-5p %c- %m%n
```


The previous three lines of code show the roll-over of the log once
per day.
```

- **Layout:** It is defined as the format of logs displayed in the log file. The appender uses the layout to format the log files (also called patterns). The highlighted code shows the pattern for the access logs:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="localhost_access_log." suffix=".txt" pattern="%h %l %u %t
"%r" %s %b" resolveHosts="false"/>
```

**Note**

```
Loggers, appenders, and layouts together help the developer to
capture the log message for the application event.
```

## Types of logging in Tomcat 8

We can enable logging in Tomcat 8 in different ways, based on the requirement. There are a total of five types of logging that we can configure in Tomcat, such as application, server, console, and so on. The following figure shows the different types of logging for Tomcat 8. These methods are used in combination with each other based on the environment needs. For example, if you have issues where Tomcat services are not displayed, then the console logs are very helpful to identify the issue, as we can verify the real-time boot sequence. Let's discuss each logging method briefly:

---
```




### Application log

These logs are used to capture the application event while running the application transaction. These logs are very useful in order to identify the application level issues. For example, suppose your application performance is slow on a particular transition, then the details of that transition can only be traced in the application log. The biggest advantage of application logs is we can configure separate log levels and log files for each application, making it very easy for the administrators to troubleshoot the application.

### Note

Log4j is used in 90 percent of cases for application log generation.

### Server log

Server logs are identical to console logs. The only advantage of server logs is that they can be retrieved anytime, but console logs are not available after we log out from the console.

### Console log

This log gives you the complete information of the Tomcat 8 startup and loader sequence. The log file is named as `catalina.out` and is found in `TOMCAT_HOME/logs`. This log file is very useful in checking the application

deployment and server startup testing for any environment. This log is configured in the Tomcat file `catalina.sh` , which can be found in `TOMCAT_HOME/bin` .

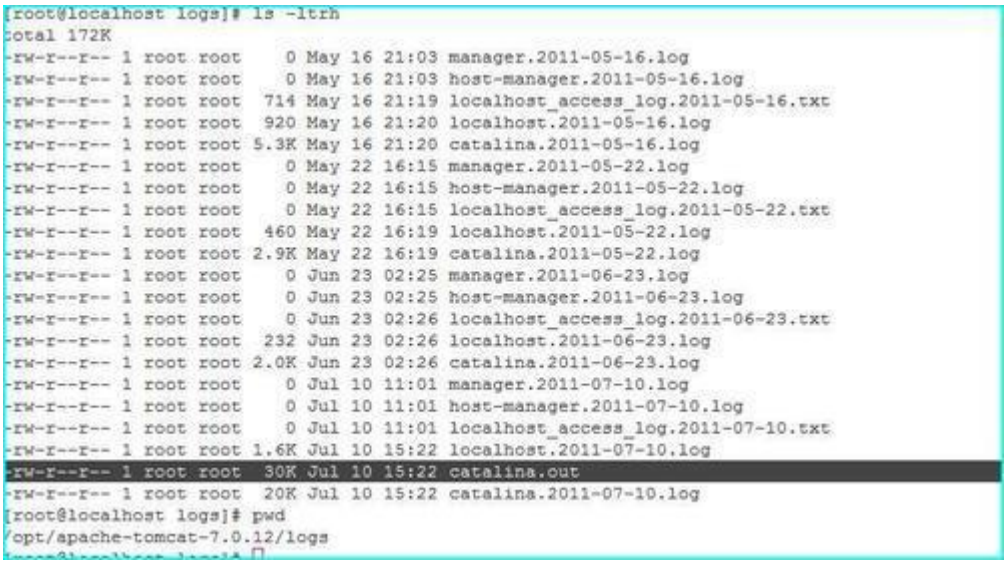
---


The previous screenshot shows the definition for Tomcat logging. By default, the console logs are configured as INFO mode.

### Note

There are different levels of logging in Tomcat such as WARN, INFO, CONFIG, and FINE. We will discuss each log level in detail in the section[ *Types of log levels in Tomcat 8*] in this lab.

---




The previous screenshot shows the Tomcat log file location, after the start of the Tomcat services.

---


The previous screenshot shows the output of the `catalina.out` file, where the Tomcat services are started in `1903 ms`.

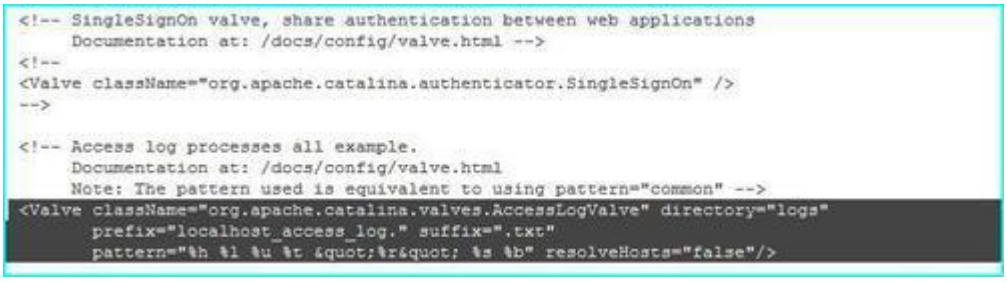
### Access log

Access logs are customized logs, which give information about the following:

- Who has accessed the application
- What components of the application are accessed
- Source IP and so on

These logs play a vital role in traffic analysis of many applications to analyze the bandwidth requirement and they also help in troubleshooting the application under a heavy load. These logs are configured in `server.xml` in `TOMCAT_HOME/conf`. The following screenshot shows the definition of the access logs. You can customize them according to the environment and your auditing requirements.

---




Let's discuss the pattern format of the access logs and understand how we can customize the logging format:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="localhost_access_log." suffix=".txt" pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s
%b" resolveHosts="false"/>
```

- **Class Name:** This parameter defines the class name used for the generation of logs. By default, Apache Tomcat 8 uses the `org.apache.catalina.valves.AccessLogValve` class for the access logs.
- **Directory:** This parameter defines the directory location for the log file. All the log files are generated in the log directory--- `TOMCAT_HOME/logs` ---but we can customize the log location based on our environment setup and then update the directory path in the definition of the access logs.

- **Prefix:** This parameter defines the prefix of the access log filename, that is, by default, the access log files are generated by the name `localhost_access_log.yy-mm-dd.txt`.
- **Suffix:** This parameter defines the file extension of the log file. Currently it is in `.txt` format.
- **Pattern:** This parameter defines the format of the log file. The pattern is a combination of values defined by the administrator, for example, `%h` = remote host address. The following screenshot shows the default log format for Tomcat 8. The access logs show the remote host address, date/time of the request, the method used for the response, URI mapping, and HTTP status code.

---

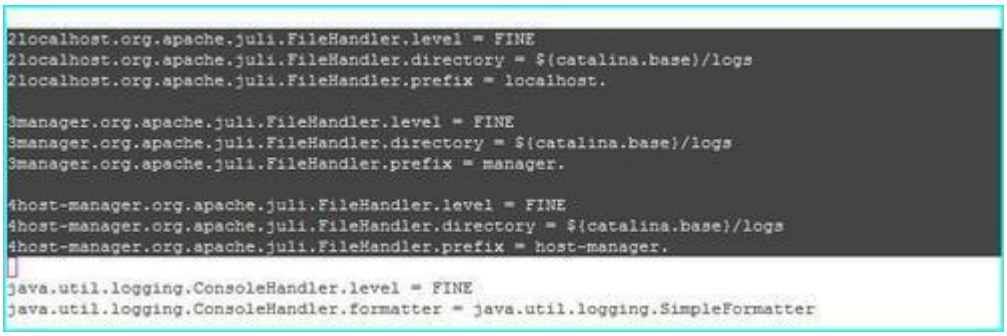

### Note

In case you have installed the web traffic analysis tool for applications, then you have to change the access logs to a different format.

### Host manager

These logs define the activity performed using the Tomcat Manager, such as the various tasks performed, the status of the application, the deployment of the application, and the lifecycle of Tomcat. These configurations are done on the `logging.properties`, which can be found in `TOMCAT_HOME/conf`.

---




The previous screenshot shows the definition of the `host`, `manager`, and `host-manager` details. If you see the definitions, it defines the log location, log level, and the prefix of the filename.

### Note

In `logging.properties`, we are defining file handlers and appenders using JULI.

The log file for `manager` looks similar to the following:

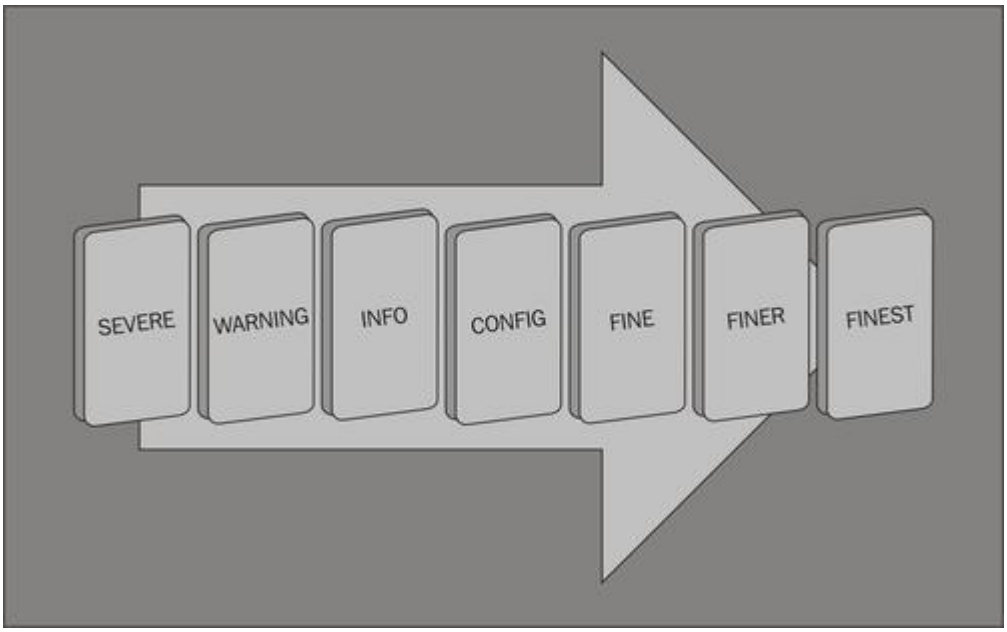
```
28 Jun, 2011 3:36:23 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
```

```
28 Jun, 2011 3:37:13 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
28 Jun, 2011 3:37:42 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: undeploy: Undeploying web application at '/sample'
28 Jun, 2011 3:37:43 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
28 Jun, 2011 3:42:59 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
28 Jun, 2011 3:43:01 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
28 Jun, 2011 3:53:44 AM org.apache.catalina.core.ApplicationContext log
INFO: HTMLManager: list: Listing contexts for virtual host 'localhost'
```

## Types of log levels in Tomcat 8

There are seven levels defined for Tomcat logging services (JULI). They can be set based on the application's requirement. The following figure shows the sequence of the log levels for JULI:

---




Every log level in JULI has its own functionality. The following table shows the functionality of each log level in JULI:

| Log level | Description |
|---|---|
| SEVERE(highest) | Captures exception and Error |
| WARNING | Warning messages |
| INFO | Informational message, related to the server activity |
| CONFIG | Configuration message |
| FINE | Detailed activity of the server transaction (similar to debug) |
| FINER | More detailed logs than FINE |
| FINEST(least) | Entire flow of events (similar to trace) |

For example, let's take an appender from logging.properties and find out the log level used; the first log appender for localhost is using FINE as the log level, as shown in the following code snippet:

```
localhost.org.apache.juli.FileHandler.level = FINE
localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
localhost.org.apache.juli.FileHandler.prefix = localhost.
```

The following code shows the default file handler configuration for logging in Tomcat 8 using JULI. The properties and log levels are mentioned:

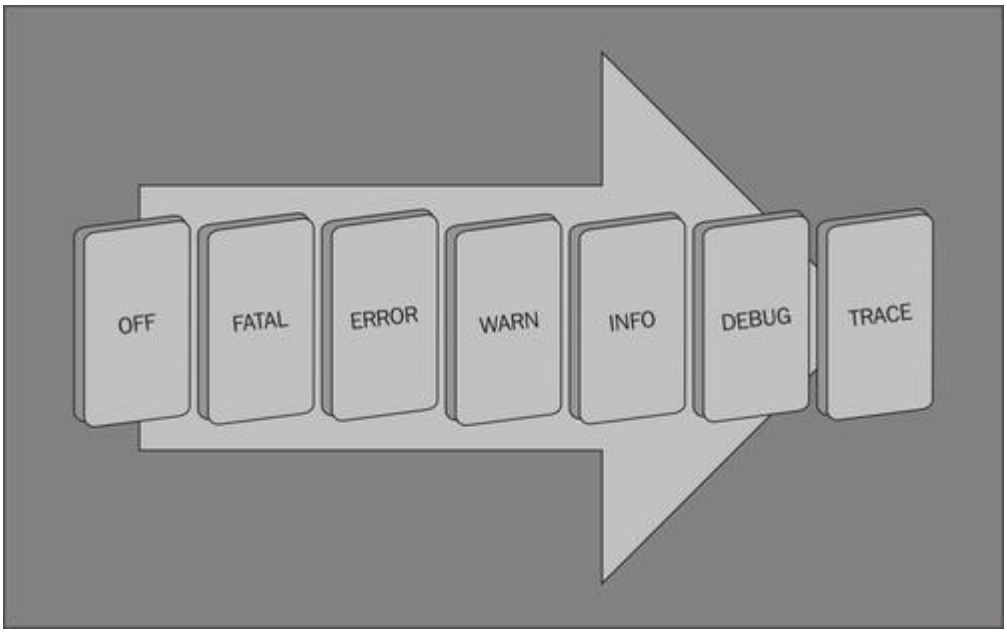
```
############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers =
2localhost.org.apache.juli.FileHandler
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager] .level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager] .handlers =
3manager.org.apache.juli.FileHandler
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host- manager].level =
INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-
manager].handlers = 4host-manager.org.apache.juli.FileHandler
```

## Log4j

Log4j is the project run by The Apache Software Foundation. This project helps in enabling the logs at the various levels of the server and application.

The major advantage of log4j is manageability. It provides the developer a freedom to change the log level at the configuration file level. Also, you can enable/disable logs at the configuration level, so there is no need to change the code. We can customize the log pattern based on the application, separately. Log4j has six log levels. The following figure shows the different types of log levels in log4j:

---




### Log level for log4j

Every log level in log4j has its own functionality. The following table shows the functionality of each log level in log4j:

| Log level | Description |
|---|---|
| OFF | This level is set when you want logging to be set as `false` (Stopped logging). |
| FATAL | This log level will print the severe errors that cause premature termination. |
| ERROR | This log level is used to capture runtime errors or unexpected conditions.<br>Expect these to be immediately visible on a status console. |
| WARN | This level is used in the previous version.<br>It gives you **almost** errors, other runtime situations that are undesirable or unexpected, but not necessarily **wrong**.<br>Expect these to be immediately visible on a status console. |
| INFO | This log level will define the interesting runtime events (startup/shutdown).<br>It is best practice to put the logs at the INFO level. |
| DEBUG | Detailed information on the flow through the system is defined in this level. |
| TRACE | This log level captures all the events in the system and application. |

#### How to use log4j

Following are the steps to be performed to use log4j:

1. Download apache-log4j-1.2.X.tar.gz from its official URL http://logging.apache.org/log4j/1.2/download.html, where X is the minor version.
2. Unzip the folder and place the log4j.jar in the lib for TOMCAT_HOME/lib and delete the juli*.jar from lib.

3. Delete the logging.properties from the TOMCAT_HOME/CONF.
4. Create a file log4j.properties in the TOMCAT_HOME/CONF and define the log appenders for the Tomcat instance. The following screenshot shows the appenders for catalina.out. Also, the highlighted code shows the roll-over of logs per day:

---

```
log4j.rootLogger=INFO, CATALINA

# Define all the appenders
log4j.appender.CATALINA=org.apache.log4j.DailyRollingFileAppender
log4j.appender.CATALINA.File=${catalina.base}/logs/catalina.
log4j.appender.CATALINA.Append=true
log4j.appender.CATALINA.Encoding=UTF-8
# Roll-over the log once per day
log4j.appender.CATALINA.DatePattern='.'dd-MM-yyyy'.log'
log4j.appender.CATALINA.layout = org.apache.log4j.PatternLayout
log4j.appender.CATALINA.layout.ConversionPattern = %d [%t] %-5p %c- %m%n

log4j.appender.LOCALHOST =org.apache.log4j.DailyRollingFileAppender
log4j.appender.LOCALHOST.File=${catalina.base}/logs/localhost.
log4j.appender.LOCALHOST.Append=true
log4j.appender.LOCALHOST.Encoding=UTF-8
log4j.appender.LOCALHOST.DatePattern='.'yyyy-MM-dd'.log'
log4j.appender.LOCALHOST.layout = org.apache.log4j.PatternLayout
log4j.appender.LOCALHOST.layout.ConversionPattern = %d [%t] %-5p %c- %m%
```

**Note**

```
You can customize the log rotation based on size, day, hour, and so
on, using the previous log4j appenders marked in bold.
```

5. Restart the Tomcat services.

**Note**

```
**Important tip for the production environment**

DEBUG and TRACE modes are ideal modes for troubleshooting, but we
have to disable them after log analysis. For a production
environment, the ideal mode is INFO (DEBUG and TRACE generate heavy
logging and also affect the server performance).

Appenders should be enabled everyday in a production environment.
This helps the administrator to perform a log analysis very easily
(the file size is less).
```

## Log level mapping

Until now, we have discussed the various log levels for JULI and log4j. Let us do a quick log level mapping for JULI and log4j. The following table shows the one-to-one mapping for log4j and JULI:

| Log level in JULI | Log level in log4j |
| --- | --- |
| SEVERE | FATAL, ERROR |
| WARNING | WARN |
| INFO | INFO |
| CONFIG | NA |
| FINE | DEBUG |
| FINER | DEBUG |
| FINEST | TRACE |

## Values for Tomcat 8

Values are defined as identifiers which change the pattern of the string in the log. Suppose you want to know the IP address of a remote host, which has accessed the website, then you add the combination of the following values mentioned in the log appenders. For example, let's customize the access logs for Tomcat 8. By default, access logs for Tomcat are defined as follows:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="localhost_access_log." suffix=".txt" pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s
%b" resolveHosts="false"/>
```

We want to change the log pattern to show the time taken to process the request. We have to add the `%T` in the patterns. The changed code is shown as follows:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="localhost_access_log." suffix=".txt"
pattern="%h %l %u %t %T &amp;quot;%r&amp;quot; %s %b" resolveHosts="false"/>
```

The following table shows the values used in Tomcat 8 for log pattern customization:

| Values | Description |
|---|---|
| %a | Remote IP address |
| %A | Local IP address |
| %b | Bytes sent, excluding HTTP headers, or '' if zero |
| %B | Bytes sent, excluding HTTP headers |
| %h | Remote hostname (or IP address if `enableLookups` for the connector is `false`) |
| %H | Request protocol |
| %l | Remote logical username from identd |
| %m | Request method (GET, POST, and so on) |
| %p | Local port on which this request was received |
| %q | Query string (prepended with a '?' if it exists) |
| %r | First line of the request (method and request URI) |
| %s | HTTP status code of the response |
| %S | User session ID |
| %t | Date and time, in Common Log format |
| %u | Remote user that was authenticated (if any) |
| %U | Requested URL path |
| %v | Local server name |
| %D | Time taken to process the request, in milliseconds |
| %T | Time taken to process the request, in seconds |
| %I | Current request thread name (can compare later with stack traces) |

## Log analysis

Log analysis is a very important and tricky issue, which needs to be handled with a lot of care. If you overlook a few lines, then you will never be able to find the root cause of the issue. Some of the best practices which need to be kept in consideration while doing the log analysis are mentioned as follows:

- Check the logs of the last 1 hour from the issue
- Always go to the first exception in the logs when the error has started
- Always keep in mind that issues are not caused due to malfunction of Tomcat, also check the other infrastructure resources

In non-DOS operating systems (Linux, Unix, Ubuntu, and so on), there are two utilities which are very useful in log analysis, `grep` and `awk`. Let's discuss `grep` and `awk` utilities briefly:

- **grep:** This utility prints the lines which match the string searched.

```
grep Error catalina.out
```

- The previous command is an example of the `grep`

```
command for searching the word \"error\" in the file
`catalina.out` and displays the lines which contain
the word \"error\".
```

- **awk:** This command is used for pattern scanning. Suppose we want to print 10 columns in the entire data file, then this command is very useful. The following screenshot shows the output of the command when run for the `/opt` directory:

```
find "location of directory " -type f -size +10000k -exec ls -lh {} \; | awk '{
print $9 ": " $5 }'
find "/opt" -type f -size +10000k -exec ls -lh {} \; | awk '{ print $9 ": " $5
}'
```

---


### Helpful commands for log analysis

Administrators are looking for shortcut commands to do their work efficiently. The following are some useful commands that I have collected during log analysis:

The following commands are used for searching big log files. Sometimes in a production environment, we get alerts for disk out of space. The following commands can be used:

- Finding large files and directories in Linux:

```
find "location of directory " type f -size +10000k -exec ls -lh {} \; | awk '{ print
$9 ": " $5 }'
```

- Finding directories with a size over 100MB:

```
find / -type d -size +100000k
```

- Sort directories as per size using `du:`

```
du --max-depth=1 -m | sort -n -r
```

- Finding directory sizes:

```
du -sh folder_name
du -ch folder_name
du -csh folder_name
```

- The following command is used for truncating huge log files on the live system (log rotation can be done without recycle of services):

```
cat /dev/null > file_name
```

The following mentioned commands are used for searching the string in different files:

- Finding ERROR exception

```
grep ERROR log_file
```

- Last 200 lines in log file:

```
tail -200 log_file
```

- Current logs to be updated

```
tail -f log_file
```

Summary

---

In this lab, we have discussed the different methods of enabling logs in Tomcat 8 using log4j and JULI. Also, we have discussed the best practices used for log analysis, tips, and tricks.

In the next lab, we will discuss the real-time issues, which are faced by the web administrator, in managing and maintaining the application production environment. So, get ready for the real fun!