BY NAGOOR BABU

LOG4J
COURSE MATERIAL
BY
NAGOOR BABU

CONTACT US:
Mobile: +91- **8885 25 26 27**
+91- **7207 21 24 27/28**
US NUM: **4433326786**

Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: **www.durgasoftonline.com**
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

BY NAGOOR BABU

SPRING LOG4J MODULE INDEX

INTRODUCTION

In Real time applications development, we may get no of problems or bugs or exceptions while executing or testing the applications, Ti identify the problems and their locations then we have to trace the applications flow of execution.

To trace applications flow of execution we will use "System.out.println(--)" at basic level.

EX:

```java
class Transaction{
void deposit(){
  System.out.println("logic for deposit");
}
void withdraw(){
  System.out.println("logic for withdraw");
}
void transfer(){
  System.out.println("logic for transfer");
}
}
class Test{
  Transaction tx = new Transaction();
  System.out.println("Before deposit() method call");
  tx.deposit()
  System.out.println("After deposit() method call");
   -----
  System.out.println("Before withdraw() method call");
  tx.withdraw();
  System.out.println("After withdraw  method call");
   ------
  System.out.println("Before transfer() method call");
  tx.transfer();
  System.out.println("After transfer() method call");
}
```

If we execute the above application then we are able to get the following output on console or command prompt.

- ✓ Before deposit() method call
- ✓ logic for deposit
- ✓ After deposit() method call
- ✓ Before withdraw() method call
- ✓ logic for withdraw

- ✓ After withdraw() method call
- ✓ Before transfer() method call
- ✓ logic for transfer
- ✓ After transfer() method call

If we use System.out.println() method in applications to trace flow of execution then we are able to get the following problems.

1. System.out.println(--) will be used for only console display, not for sending data to any other output systems like file systems[html files, xml files, text files...], databases, network,....
2. System.out.println() method contains synchronized block to display data, it is heavy weight, expensive and time consuming.
3. In Applications, it is not suggestible to write too many no of System.out.println() methods in applications, because, it will reduce application performance.
4. System.out.println() method is usefull in Development environment only, It is not suitable in production environment , because, if we use System.out.println() method in server side applications then it will display messages in servers console only, it will not be avaiable to users.
5. System.out.println() will display the messages on console or on command prompt, it will not show differences in Error messages, warning messaages, normal information,....
6. System.out.println() is suitable for simple standalone applications, not for complex enterprise applications.

To overcome all the problems while tracing applications we have to use Logging.

Logging : It is the process of writing log messages during the execution of a program to a central place. This logging allows you to report and persist error and warning messages as well as info messages so that the messages can later be retrieved and analyzed.

In general, in java applications, Logging is required

1. To understand flow of execution in applications
2. To manage exception messages when Exceptions are occurred in java applications
3. To manage the event - notification messages in file systems....

Logging Framework: It is a set of classes and interfaces or a product to perform Logging in Java applications.

EX:

1. Java provided java.util Logging
2. Log4j
3. LogBack
4. Commons Logging

5. ObjectGuy
6. TinyLog

Advantages of Logging Frameworks:

1. Problem diagnosis: In ingeneral, in application development, we are able to detect the problems which are occurred in compilation time or in execution, but, some hidden problems are existed inside the applications, which may not come in out side direcly, but, they will give effect to our application execution, in this situation, if we use good Logging Framework then it is possible to detect the hidden problems quickly and we can fix them easily.

2. Quick Debugging: Once if we diagnose the problem and if we identify the location of the problem by using logging framework then it is very simple to fix that problem and it simplifies Debugging and it able to reduce overall Debugging time.

3. Easy Maintenance: If we provide good Logging Framework to perform logging in our applications then it will provide somde description or information about our application which is usefull to manage our applications easily.

4. History: In general, all Logging Frameworks are tracing the applications flow of execution and they are able to store tracing details in a file system, this logging details are usefull to analyze the problems and to solve the problems.

5. Cost and Time Saving: Logging Frameworks will simplifies debugging, easy maintenance , persisting application information,..... , these qualities of Logging Framework saves time and cost of the application.

Drawbacks with Logging Frameworks:

1. Logging Frameworks needs to write some extra code in our java applications, it will increase overhead to the application.
2. Logging Frameworks will provide some extra code in our applications, it will increase application execution time at runtime, it will reduce application performance.
3. Logging Framework will provide some extra code in application, so that, it will increase application length.
4. Poor logging strategies will increase confusion to the developers.

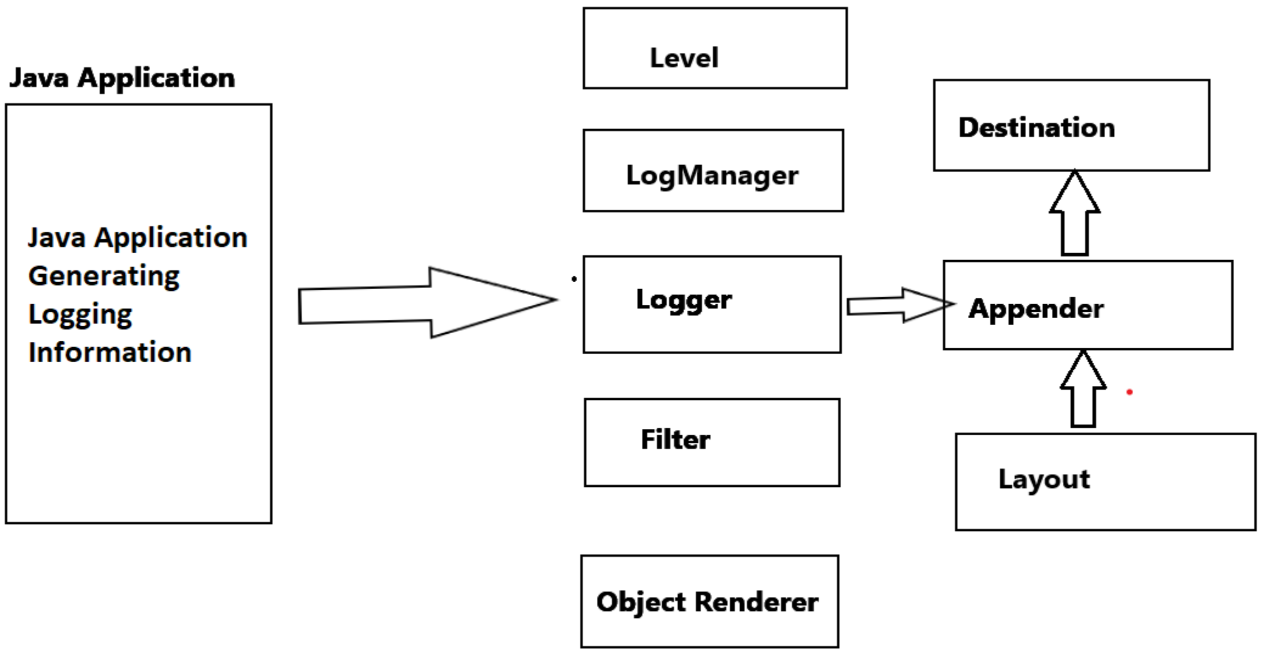
Log4j

Log4J is a reliable, fast and flexible framework written in JAVA to perform logging in Java applications and it is provided by Apache Software Foundations.

Log4j Features:

- Log4j is able to allow more than one thread at a time with out providing data inconsistency ,so that, Log4j is thread-safe.
- Log4j will not slow down the application execution, it will be optimized to improve application performance.
- Log4j is providing environment to send logging messages to more than one output appenders .
- Log4j supports internationalization.
- Log4j is providing services for both predefined and user defined facilities, it able to provide some customizations also.
- Log4j allows to set logging behaviours at runtime through the configuration file.
- Log4j is providing very good environment to traace Exceptions from its root.
- Log4j uses multiple levels like ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL to generate messages.
- Log4j allows to change the format of log output by extending Layout class.
- Log4j is using appenders to generate log messages.

Log4j Architecture:



Log4j Arch contains mainly the following Objects:

1. Logger
2. Appender
3. Layout
4. Level
5. LogManager
6. Filter
7. ObjectRender

1. Logger:
This Object is responsibile to get logging messages from Java applications

2. Appender:
The main intention of Appender object is to get logging messages from Logger object and publishing that logging messages to the preffered destinations. Each and Every Appender Object must have atleast one Destination object inorder to send logging messages.

EX: ConsoleAppender is able to store logging messages on Console.

3. Layout:
The main intention of Layout object is to format logging messages in different styles.Layout Object is used by Appender object just before publishing Logging Messages.

4. Level:
The main intention of Level object is to define granualarity and priority of any Logging information. Each and every Logging information must be with a particular Logging Level.

Log4j API has provided the following Levels to the Logging messages.

1. ALL
2. TRACE
3. DEBUG
4. ENFO
5. WRAN
6. ERROR
7. FATAL
8. OFF

Log4j is giving priorities for the logging messages in the following order.
ALL >TRACE > DEBUGG > INFO > WARN > ERROR > FATAL>OFF

6. LogManager

LogManager is the central component , it able to manage Log4j framework, it will read all the initial configuration details from the configuration file and stored the Logger objects in namespace hierarchy with a particular reference name for futer reference. If our application access any Logger object on the basis of the reference name then LogManager will return the existed Logger object otherwise it will create new Logger object and return to the application.

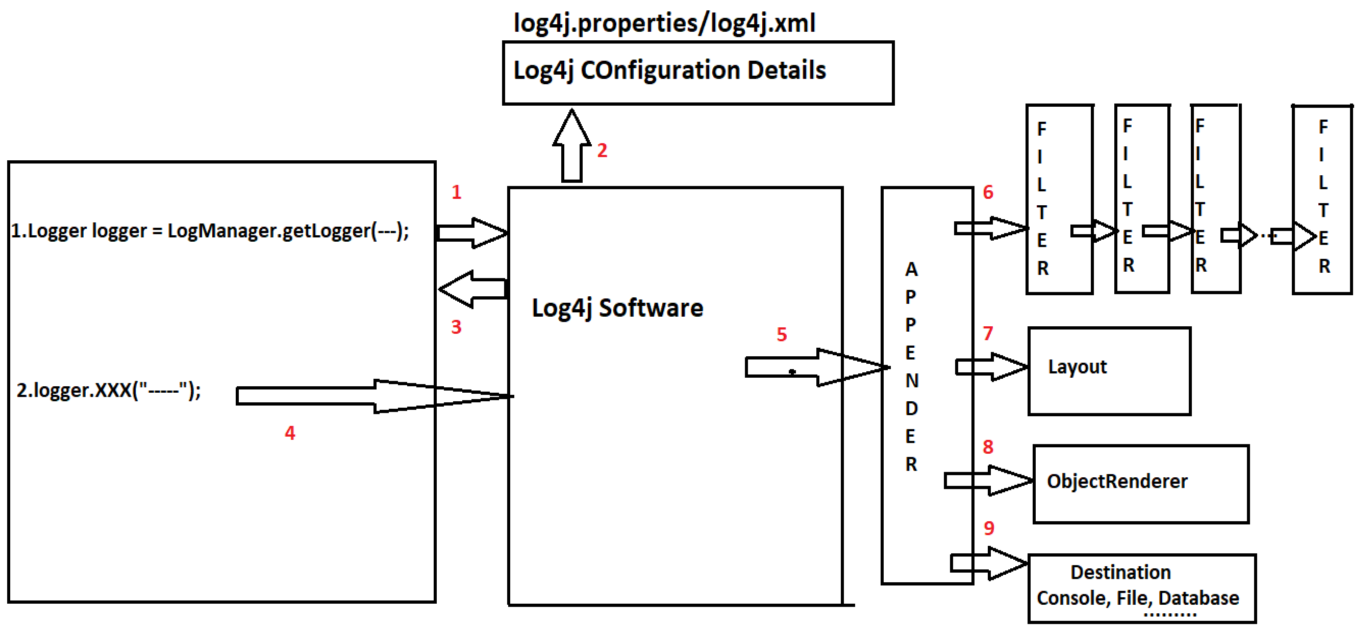
7. Filter:

The main intention of Filter object is used to analyze logging information and takes the decision whther the messages must be logged or not.
An Appender object may have no of Filter objects, they must approve our logging message before publishing logging messages in destination.

8.ObjectRender

This Object will be used by Layout object in order to get string representation of the several objects which are passed through Log4j framework.

Log4j Internal Flow:



1. If we create Logger object in Java application then a request will be send to Log4j Software about to create Logger object.

2. Log4j Software will search for log4j.properties/log4j.xml file in classpath, if it is existed then Log4j Software will load and parse log4j configuration file then Log4j software will create Logger object.
3. After creating Logger object, Log4j Software will return Logger object to Java Application.
4. If we access any logger method from Logger reference then Java application will send logging responsibility to Log4j software.
5. Log4j Software will activate Appender object inorder to perform logging.
6. Appender will recognize all the filters which are associated then Appender will execute all the filters in sequence, where filters will filters the logging messages to persist or display.
7. Appender will activate Layout object , it will apply the layout styles on the logging messages.
8. After Styling Logging messages, Appender will activate Object Renderer to recognize the destination.
9. Sending Logging messages to the respective Destination.

Log4j Configuration File:

Log4j Configuration File includes all Log4j configuration details like rootLogger, Layout configurations, Appenders configuration,.

In Log4j , we are able to use the following two types of configuration files.

1.log4j.properties
2.log4j.xml

IN both the configuratin files, we have to use the following configurations at basic java applications.

1. Declare rootLogger and initialize rootLogger
2. Declare appender
3. Declare Layout and its conversionPattern.

EX: log4j.properties:

✓ log4j.rootLogger =TRACE, CONSOLE
✓ log4j.appender.CONSOLE = org.apache.log4j.ConsoleAppender
✓ log4j.appender.CONSOLE.layout = org.apache.log4j.PatternLayout
✓ log4j.appender.CONSOLE.layout.conversionPattern=%d{yyyy-mm-dd hh:mm:ss } : This is %m%n

- Where "log4j.rootLogger" property will take Logging LEVEL and appender name,
- where Log4j framework will display all the messages which are same as the specified Log level and lower than the specified Log level.

- Where appender name is a name which is referring a particular Appender.

EX: log4j.rootLogger = TRACE, CONSOLE

Where "log4j.appender.CONSOLE" property will take a particular Appender like ConsoleAppender, FileAppender,...

EX: log4j.appender.CONSOLE = org.apache.log4j.ConsoleAppender

Where "log4j.appender.CONSOLE.layout" property will configure Layout configfuration.

EX: log4j.appender.CONSOLE.layout = org.apache.log4j.PatternLayout

Where PattrenLayout needs a property like conversionPattern to take a particular pattern to display messages.

EX: "log4j.appender.CONSOLE.layout.conversionPattern = %m%n

Note: Where %m%n in the sense, %m is to display message and %n is to bring cursor to next line.

log4j.xml Configuration:

In XML confioguration we have to use the following configuration.

1. Appender configuration.
2. Layout Configuration
3. Root Logger COnfiguration

To provide XML configuration then we have to use the following xml tags in log4j.xml file.

log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
   debug="false">

<appender name="console" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{DD/MM/YYYY HH:mm:SS} %-
5p %c{1} - %m%n" />
   </layout>
</appender>
```

```
<root>
   <priority value="ERROR" />
   <appender-ref ref="console" />
</root>

</log4j:configuration>
```

- ✓ Where <log4j:configuration> isa a root tag, it able to take log4j configuration details.
- ✓ Where <appender> tag is able to configure Appender class with a logical name.
- ✓ Where "name" attribute in <appender> tag will take logical name to the appender.
- ✓ Where "class" attribute in <appender> tag is able to take fully qualified name of the Appender class, that is, "org.apache.log4j.ConsoleAppender"
- ✓ Where <layout> tag can be used to configure Layout class.
- ✓ Where "class" attribute in <layout> tag will take fully qualified name of Layout class, that is, "org.apache.log4j.PatternLayout".
- ✓ Where <param> tag in <layout> tag will take layout parameter .
- ✓ Where "name" attribute will take parameter name, tyhat is , "ConversionPattern".
- ✓ Where "value" attribute will take value of the parameter , that is, %d{DD/MM/YYYY HH:mm:SS} %-5p %c{1} - %m%n
- ✓ Where <root> tag will provide rootLogger configuration.
- ✓ Where <priority> tag tag will take logger level with "value" attribute.
- ✓ Where <appender-ref> tag will take logical name of the appender with "ref" attribute.

Steps to Use Log4j in Java Application:

1. Download log4j-1.2.17.zip from "https://logging.apache.org/log4j/1.2/download.html"
2. Create Java project in Eclipse IDE and add log4j1.2.17.jar file in build path
3. Create log4j configuration file[log4j.properties or log4j.xml].
4. Create Java class.
5. Run Java project [Suggestible in Debug mode].

Example:

log4j.properties

- ✓ log4j.rootLogger =FATAL, CONSOLE
- ✓ log4j.appender.CONSOLE = org.apache.log4j.ConsoleAppender
- ✓ log4j.appender.CONSOLE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.CONSOLE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } : This is %m%n

Log4jTest.java

```java
package log4japp1;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {

   public static void main(String[] args) {
      Logger logger = LogManager.getLogger(Log4jTest.class);
      logger.trace("trace Message");
      logger.debug("debug Message");
      logger.info("info Message");
      logger.warn("warn Message");
      logger.error("Error Message");
      logger.fatal("fatal Message");

   }
}
```

Note: If we want to use XML configuration then we have to use the following XML file
log4j.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
   debug="false">

<appender name="console" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{DD/MM/YYYY HH:mm:SS} %-5p %c{1} - %m%n" />
   </layout>
</appender>

<root>
   <priority value="ERROR" />
   <appender-ref ref="console" />
</root>

</log4j:configuration>
```

Appenders

Appenders:

Appender is an object, it can be used to get logging messages from Logger object and publishing that logging messages to the preffered destinations. Each and Every Appender Object must have atleast one Destination object inorder to send logging messages.

Log4j has provided the following appenders inorder to publish logging messages.

1. ConsoleAppender
2. FileAppender
3. JDBCAppender
4. JMSAppender
5. SocketAppender
6. SMTPAppender
7. AsyncAppender
8. AppenderSkeleton
9. DailyRollingFileAppender
10. ExternallyRolledFileAppender
11. LF5Appender
12. NTEventLogAppender
13. NullAppender
14. RollingFileAppender
15. SocketHubAppender
16. SyslogAppender
17. TelnetAppender
18. WriterAppender

If we want to manage all logging messages in a File then we have to use FileAppenders.

There are Four types of File Appenders.

1. FileAppender
2. RollingFileAppender
3. DailyRollingFileAppender
4. ExternallyRolledFileAppender

1. FileAppender:

If we want to use FileAppender in Java applications then we have to use the following properties in log4j.properties file.

1. log4j.rootLogger
2. log4j.appender.Ref_Name
3. log4j.appender.Ref_Name.File : It will name and location of the file to manage logging messages.
4. log4j.appender.Ref_Name.layout
5. log4j.appender.Ref_Name.layout.ConversionProperty

EX:

log4j.properties

- ✓ log4j.rootLogger =ALL, FILE
- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.File=./logging/logs.log
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } : This is %m%n

Log4jTest.java

```java
package log4japp1;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {

    public static void main(String[] args) {

        Logger logger = LogManager.getLogger(Log4jTest.class);
        logger.trace("trace Message");
        logger.debug("debug Message");
        logger.info("info Message");
        logger.warn("warn Message");
        logger.error("Error Message");
        logger.fatal("fatal Message");

    }
}
```

If we want to use both ConsoleAppender and FileAppender in Java Applications then we have to define two Ref_Names at 'log4j.rootLogger' property.

EX:
log4j.properties

```
log4j.rootLogger =ALL, CONSOLE, FILE
```

#ConsoleAppender Configuration

- ✓ log4j.appender.CONSOLE = org.apache.log4j.ConsoleAppender
- ✓ log4j.appender.CONSOLE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.CONSOLE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } %-c %p : This is %m%n

#FileAppender Configuration

- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.File=./logging/logs.log
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } %-c %p : This is %m%n

Log4jTest.java

```java
package log4japp1;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {

    public static void main(String[] args) {

        Logger logger = LogManager.getLogger(Log4jTest.class);
        logger.trace("trace Message");
        logger.debug("debug Message");
        logger.info("info Message");
        logger.warn("warn Message");
        logger.error("Error Message");
        logger.fatal("fatal Message");

    }
}
```

2. RollingFileAppender

The main intention of RollingFileAppender is to manage logging messages to the other new files when the present log file is getting exceded as per the file size limit.

If we want to use RollingFileAppender in log4j applications then we have to use "org.apache.log4j.RollingFileAppender" class and we have to use all the FileAppender properties and the following extra properties.

1. log4j.appender.Ref_Name.MaxFileSize : It will take max File size in the form of KB
2. log4j.appender.Ref_Name.MaxBackupIndex= It able to create maximum backup files.

EX:

log4j.properties

log4j.rootLogger =ALL, FILE

#FileAppender Configuration

- ✓ log4j.appender.FILE = org.apache.log4j.RollingFileAppender
- ✓ log4j.appender.FILE.file= e:/loggers/logs.txt
- ✓ log4j.appender.FILE.MaxFileSize= 2kb
- ✓ log4j.appender.FILE.MaxBackupIndex= 3
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } %-c %p : This is %m%n

Log4jTest.java

```
package log4japp1;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
   static Logger logger = LogManager.getLogger(Log4jTest.class);
   public static void main(String[] args) {

      logger.trace("trace Message");
      logger.debug("debug Message");
      logger.info("info Message");
      logger.warn("warn Message");
      logger.error("Error Message");
```

```
        logger.fatal("fatal Message");

    }
}
```

3. DailyRollingFileAppender

The main intention of DailyRollingFileAppender is to roll files in FileAppender on daily basis.

If we want to use DailyRollingFileAppender in log4j applications then we have to use "org.apache.log4j.DailyRollingFileAppender" and we have to use "datePattern" property in properties file.

Note: If we are using DailyRollingFileAppender in our application then it is not required to use "maxFileSize" and "maxBackupIndex" properties in log4j.properties file.

Note: The default pattern for datePattern property is '.' '.' yyyy-MM-dd , it will roll over every day midnight

For "datePattern" property we are able to use the following values.

1. '.' yyyy-MM : It will roll over at the end of each month and at the beginning of the next month.
2. '.' yyyy-MM-dd : It will roll over at midnight each day.
3. '.' yyyy-MM-dd-a :It will roll over at midday and midnight of each day.
4. '.' yyyy-MM-dd-HH : It will roll over at the top of every hour.
5. '.' yyyy-MM-dd-HH-mm : It will roll over every minute.
6. '.' yyyy-ww : It will roll over on the first day of each week depending upon the locale.

Example:

log4j.properties

log4j.rootLogger =ALL, FILE

#FileAppender Configuration

- ✓ log4j.appender.FILE = org.apache.log4j.DailyRollingFileAppender
- ✓ log4j.appender.FILE.file= e:/loggers/logs.txt
- ✓ log4j.appender.FILE.DatePattern='.' yyyy-MM-dd-HH-mm
- ✓ #log4j.appender.FILE.MaxFileSize= 2kb
- ✓ #log4j.appender.FILE.MaxBackupIndex= 3
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{yyyy-mm-dd hh:mm:ss } %-c %p : This is %m%n

Log4jTest.java

```
package log4japp1;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
    static Logger logger = LogManager.getLogger(Log4jTest.class);
    public static void main(String[] args) {


        logger.trace("trace Message");
        logger.debug("debug Message");
        logger.info("info Message");
        logger.warn("warn Message");
        logger.error("Error Message");
        logger.fatal("fatal Message");

    }
}
```

4. ExternallyRolledFileAppender:

This appender listens on a socket on the port specified by the Port property for a "RollOver" message. When such a message is received, the underlying log file is rolled over and an acknowledgment message is sent back to the process initiating the roll over.

Note: ExternallyRolledFileAppender is required Socket Programming and it is deprecated

Layouts

Layouts:

The main intention of Layout is to define a partidular Structer for the logging messages.

There are four types of Layouts existed in Log4j.

1. SimpleLayout
2. PatternLayout
3. HTMLLayout
4. XMLLayout

1. Simple Layout:

SimpleLayout is able to prepare logging messages with "LEVEL - Log_Message" . Log4j has provided a predefiend class to represent Simple Layout,that is, org.apache.log4j.SimpleLayout

EX:

log4j.properties

```
log4j.rootLogger = ALL, FILE
log4j.appender.FILE = org.apache.log4j.FileAppender
log4j.appender.FILE.file = E:/loggers/logs.log
log4j.appender.FILE.layout = org.apache.log4j.SimpleLayout
```

Test.java

```java
package com.durgasoft.log4j;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
    static Logger logger = LogManager.getLogger(Log4jTest.class);
    public static void main(String[] args) {
        logger.trace("Trace Message");
        logger.debug("Debug Message");
        logger.info("Info Message");
        logger.warn("Warn Message");
        logger.error("Error Message");
        logger.fatal("Fatal Message");

    }
```

```

}
```

OP:

- TRACE - Trace Message
- DEBUG - Debug Message
- INFO - Info Message
- WARN - Warn Message
- ERROR - Error Message
- FATAL - Fatal Message

2. Pattern Layout:

It able to prepare logging messages w.r.t the provided pattern. To represent Pattern Layout, Log4j has provided a predefined class in the form of "org.apache.log4j.PatternLayout".

To define pattern for the logging messages, we must use "ConversionPattern" property from PatternLayout class.

EX:

log4j.properties

- ✓ log4j.rootLogger = ALL, FILE
- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.file = E:/loggers/logs.log
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{dd/MM/YYYY HH:mm:SS} %-c %p : This Is %m%n

Log4jTest.java

```java
package com.durgasoft.log4j;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
    static Logger logger = LogManager.getLogger(Log4jTest.class);
    public static void main(String[] args) {
        logger.trace("Trace Message");
        logger.debug("Debug Message");
        logger.info("Info Message");
        logger.warn("Warn Message");
        logger.error("Error Message");
        logger.fatal("Fatal Message");

    }

}
```

OP:

- ✓ 22/08/2018 18:50:885 com.durgasoft.log4j.Log4jTest TRACE : This Is Trace Message
- ✓ 22/08/2018 18:50:886 com.durgasoft.log4j.Log4jTest DEBUG : This Is Debug Message
- ✓ 22/08/2018 18:50:886 com.durgasoft.log4j.Log4jTest INFO : This Is Info Message
- ✓ 22/08/2018 18:50:886 com.durgasoft.log4j.Log4jTest WARN : This Is Warn Message
- ✓ 22/08/2018 18:50:886 com.durgasoft.log4j.Log4jTest ERROR : This Is Error Message
- ✓ 22/08/2018 18:50:886 com.durgasoft.log4j.Log4jTest FATAL : This Is Fatal Message

Note: We will use these patterns in general in PatternLayout for ConversionPattern property.

- %d ----> Date
- %c ----> Class to which we are providing Logging
- %P ----> Logging Level
- %L ----> Line NUmber
- %m ----> Logging Message
- %n ----> New Line Character

3. HTML Layout

It able to provide all logging messages in the form of Html file. To represent this Layout Log4j has provided a predefined class in the form of **"org.apache.log4j.HTMLLayout"**

EX:

log4j.properties

- ✓ log4j.rootLogger = ALL, FILE
- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.file = E:/loggers/logs.html
- ✓ log4j.appender.FILE.layout = org.apache.log4j.HTMLLayout

Log4jTest.java

```
package com.durgasoft.log4j;
```

```java

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
    static Logger logger = LogManager.getLogger(Log4jTest.class);
    public static void main(String[] args) {
        logger.trace("Trace Message");
        logger.debug("Debug Message");
        logger.info("Info Message");
        logger.warn("Warn Message");
        logger.error("Error Message");
        logger.fatal("Fatal Message");
    }
}
```

4. XML Layout

It able to provide logging messages in the form of XML document. To rperepsent XMLLayout, Log4j has provided a predefined class in the form of **"org.apache.log4j.xml.XMLLayout"**

EX:

log4j.properties

```
log4j.rootLogger = ALL, FILE
log4j.appender.FILE = org.apache.log4j.FileAppender
log4j.appender.FILE.file = E:/loggers/logs.xml
log4j.appender.FILE.layout = org.apache.log4j.xml.XMLLayout
```

Log4jTest.java

```java
package com.durgasoft.log4j;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Log4jTest {
    static Logger logger = LogManager.getLogger(Log4jTest.class);
    public static void main(String[] args) {
        logger.trace("Trace Message");
        logger.debug("Debug Message");
        logger.info("Info Message");
        logger.warn("Warn Message");
        logger.error("Error Message");
        logger.fatal("Fatal Message");
    }
}
```

Log4j in JDBC Applications:

log4j.properties

- ✓ log4j.rootLogger = ALL, FILE
- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.file = E:/loggers/logs.txt
- ✓ log4j.appender.FILE.append = false
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{dd/MM/YYYY HH:mm:SS} %-c %p %L : %m%n

Test.java

```java
package com.durgasoft.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Test {
    //static Logger logger = LogManager.getLogger(Test.class);
    static Logger logger = Logger.getLogger(Test.class);
    public static void main(String[] args) {
        logger.info("Application Starting Point");
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            logger.info("Driver Loaded");
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system
", "durga");
            if(con == null) {
                logger.warn("Connection Not Created ");
```

```
            }else {
                logger.info("Connection Established :"+con);
            }
            st = con.createStatement();
            if(st == null) {
                logger.warn("Statement is not created");
            }else {
                logger.info("Statement is Created :"+st);
            }
            String query = "select *emp1";
            rs = st.executeQuery(query);
            if(rs == null) {
                logger.warn("ResultSet is not Created with '"+query+"'");
            }else {
                logger.info("ResultSet Object is Created :"+rs);
            }
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("-----------------------------");
            while(rs.next()) {
                System.out.print(rs.getInt("ENO")+"\t");
                System.out.print(rs.getString("ENAME")+"\t");
                System.out.print(rs.getFloat("ESAL")+"\t");
                System.out.println(rs.getString("EADDR"));
            }
        } catch (Exception e) {
            e.printStackTrace();
            logger.error("Exception :"+e.toString());
        }finally {
            try {

                rs.close();
                st.close();
                con.close();
                logger.info("Closing all Resources");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        logger.info("End of main(");
    }
}
```

Log4j in Web Applications:

1. Keep Log4j1.2.7.jar in web application lib folder.
2. Prepare Configuration File under src folder / under classes folder.
3. It is suggestible to use FileAppenders.
4. In Servlets generate Loggers.

Example:

loginform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Durga Software Solutions</h2>
<h3>User Login Page</h3>
<form  method="POST" action="./login">
<table>
<tr>
    <td>User Name</td>
    <td><input type="text" name="uname"/></td>
</tr>
<tr>
    <td>Password</td>
    <td><input type="password" name="upwd"/></td>
</tr>
<tr>
    <td><input type="submit" value="Login"/></td>
</tr>
</table>
</form>
</body>
</html>
```

success.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Durga Software Solutions</h2>
<h3>User Login Status</h3>
<h2>
Login Success
</h2>
<h3>
<a href="./loginform.html">|Login Page|</a>
</h3>
</body>
</html>
```

failure.html

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Durga Software Solutions</h2>
<h3>User Login Status</h3>
<h2>
Login Failure
</h2>
<h3>
<a href="./loginform.html">|Login Page|</a>
</h3>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.c
```

```
om/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" id="WebApp_ID" version="2.5">
 <display-name>app4</display-name>
 <welcome-file-list>
   <welcome-file>index.html</welcome-file>
   <welcome-file>index.htm</welcome-file>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>default.html</welcome-file>
   <welcome-file>default.htm</welcome-file>
   <welcome-file>default.jsp</welcome-file>
 </welcome-file-list>
 <servlet>
   <description></description>
   <display-name>LoginServlet</display-name>
   <servlet-name>LoginServlet</servlet-name>
   <servlet-class>com.durgasoft.servlets.LoginServlet</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>LoginServlet</servlet-name>
   <url-pattern>/login</url-pattern>
 </servlet-mapping>
</web-app>
```

log4j.properties

- ✓ log4j.rootLogger = ALL, FILE
- ✓ log4j.appender.FILE = org.apache.log4j.FileAppender
- ✓ log4j.appender.FILE.file = E:/loggers/logs.txt
- ✓ #log4j.appender.FILE.datePattern = '.'dd-MM-yyyy-HH-mm
- ✓ log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ #log4j.appender.FILE.layout = org.apache.log4j.PatternLayout
- ✓ log4j.appender.FILE.layout.conversionPattern = %d{dd/MM/YYYY HH:mm:SS} %-C %p %L : %m%n

LoginServlet.java

```java
package com.durgasoft.servlets;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```java
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.Logger;

import com.durgasoft.service.UserService;
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public static Logger logger = Logger.getLogger(LoginServlet.class);
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        logger.info("LoginServlet Started-doPost()");
        String uname = request.getParameter("uname");
        String upwd = request.getParameter("upwd");
        logger.info("User Request Recieved with User Name :"+uname+" With password :"+upwd);

        UserService userService = new UserService();
        String status = userService.checkLogin(uname, upwd);

        RequestDispatcher reqDispatcher = null;
        if(status.equals("success")) {
            reqDispatcher = request.getRequestDispatcher("success.html");
            reqDispatcher.forward(request, response);
        }else {
            reqDispatcher = request.getRequestDispatcher("failure.html");
            reqDispatcher.forward(request, response);
        }
        logger.info("Login Status :"+status);
        logger.info("End of LoginServlet");
    }
}
```

UserService.java

```java
package com.durgasoft.service;
import org.apache.log4j.Logger;

import com.durgasoft.servlets.LoginServlet;

public class UserService {
    static Logger logger = Logger.getLogger(UserService.class);
    String status = "";
```

```
    public String checkLogin(String uname, String upwd) {


        logger.debug("UserService : checkLogin() Method started");
        if(uname.equals("durga") && upwd.equals("durga")) {
            logger.info("UserService :Login Success");
            status = "success";
        }else {
            logger.info("USerService :Login Failure");
            status = "failure";
        }
        logger.info("USerService :End of CheckLogin()");
        return status;
    }
}
```