**Log4j**

Contents

[What Is Log4j, Why Log4j Came Into Picture 2](#_Toc502688304)

[What Are The Main Components Of Log4J 2](#_Toc502688305)

[Logger 2](#_Toc502688306)

[Getting Logger Object 2](#_Toc502688307)

[Example 2](#_Toc502688308)

[Priority Order 3](#_Toc502688309)

[Appender 3](#_Toc502688310)

[Layout 4](#_Toc502688311)

[Log4j Hello World Program 4](#_Toc502688312)

[Files Required 4](#_Toc502688313)

[Directory Structure 4](#_Toc502688314)

[my.txt 5](#_Toc502688315)

[Explanation 5](#_Toc502688316)

[How To Create Log4j.properties File 5](#_Toc502688317)

[log4j.properties 5](#_Toc502688318)

[Example On Log4j.properties File With FileAppender & SimpleLayout 5](#_Toc502688319)

[Files Required 6](#_Toc502688320)

[Directory Structure 6](#_Toc502688321)

[log4j.properties 6](#_Toc502688322)

[my.txt 6](#_Toc502688323)

[Execution Flow 6](#_Toc502688324)

[Explanation 7](#_Toc502688325)

[Log4j Example On Using FileAppender And ConsoleAppender Simultaneously 7](#_Toc502688326)

[Files Required 7](#_Toc502688327)

[Directory Structure 7](#_Toc502688328)

[log4j.properties 7](#_Toc502688329)

[my.txt 8](#_Toc502688330)

# What Is Log4j, Why Log4j Came Into Picture

While developing Java/J2EE applications, for debugging an application that is to know the status of a java application at its execution time, in general we use system.out.println statements in the application right…

But we have some disadvantages while using SOPL (system.out.println) statements.

* Generally SOPL statements are printed on console, so there are temporary messages and when ever the console is closed then automatically the messages are removed from the console
* It is not possible to store the SOPL messages in a permanent place and these are single threaded model, means these will prints only one by one message on the console screen
* In order to overcome the problems  of  SOPL statements Log4j came into picture, with Log4j we can store the flow details of  our Java/J2EE in a file or databases
* This is a Open Source tool given by Apache, for only java projects, to record or write the status of an application at various places
* Working with log4j is nothing but working with classes & interfaces given in org.apache.log4j.\*
* Log4j is a common tool, used for small to large scale Java/J2EE projects
* In Log4j we use log statements rather SOPL statements in the code to know the status of a project while it is executing
* In real time, after a project is released and it is installed in a client location then we call the location as on-site right, when executing the program at on-site location, if we got any problems occurred then these problems must report to the off showered engineers,  in this time we used to mail these Log files only so that they can check the problems easily

# What Are The Main Components Of Log4J

We have mainly 3 components to work with Log4j …

* Logger
* Appender
* Layout

## Logger

* Logger is a class, in org.apache.log4j.\*
* We need to create Logger object one per java class
* This component enables Log4j in our java class
* Logger methods are used to generate log statements in a java class instead of sopls
* So in order to get an object of Logger class, we need to call a static factory method [ factory method will gives an object as return type ]
* We must create Logger object right after our class name, i will show you

## Getting Logger Object

static Logger log = Logger.getLogger(YourClassName.class.getName())

Note:  while creating a Logger object we need to pass either fully qualified class name or class object as a parameter, class means current class for which we are going to use Log4j.

## Example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class Client {       static Logger l = Logger.getLogger(Client.class.getName());       public static void main(String[] args) {       // Our logic will goes here     }  } |

Logger object has some methods, actually we used to print the status of our application by using these methods only

We have totally 5 methods in Logger class

* debug()
* info()
* warn()
* error()
* fatal()

As a programmer its our responsibility to know where we need to use what method, did you observe there ? method names are different right, but all are same :-)

## Priority Order

debug < info < warn < error < fatal

I mean, fatal is the highest error like some database down/closed

Remember:  Friends don’t confuse by seeing all these **5** methods all are same, for example if our application is about 100 lines and we have JDBC related code in some 45th line or some where there we used to write fatal() method.  All it could be is just for human identification purpose names are different, else these 5 methods will print one text message only ;)

You will get more clarity once you saw the first program on log4j.

## Appender

Appender job is to write the messages into the external file or database or smtp

* Logger classes generates some statements under different levels right, this Appender takes these logstatements and stores in some files or database
* Appender is an interface

In log4j we have different Appender  implementation classes

* FileAppender [ writing into a file ]
* ConsoleAppender [ Writing into console ]
* JDBCAppender [ For Databases ]
* SMTPAppender [ Mails ]
* SocketAppender [ For remote storage ]
* SocketHubAppender
* SyslogAppendersends
* TelnetAppender

Again in FileAppender we have 2 more

* RollingFileAppender
* DailyRollingFileAppender

For now just remember, i will explain while executing the program

## Layout

This component specifies the format in which the log statements are written into the destinationrepository by the appender

We have different type of layout classes in log4j

* SimpleLayout
* PatternLayout
* HTMLLayout
* XMLLayout

So will see one simple example on Log4j… [ In The Next Session ]

# Log4j Hello World Program

Let us see one simple program in Log4j

For working with log4j, we must set log4j.jar in our class path

## Files Required

* Client.java
* my.txt   [ We will let the appender to write into this file ]

## Directory Structure

Client.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import org.apache.log4j.Appender;  import org.apache.log4j.FileAppender;  import org.apache.log4j.Layout;  import org.apache.log4j.Logger;  import org.apache.log4j.SimpleLayout;    public class Client {      static Logger l = Logger.getLogger(Client.class.getName());      public static void main(String[] args) {      Layout l1 = new SimpleLayout();    Appender a;    //Appender a = new ConsoleAppender(l1);    try    {    a = new FileAppender(l1,"my.txt", false);      // 3rd parameter is true by default    // true = Appends the data into my.txt    // false = delete previous data and re-write      l.addAppender(a);    }    catch(Exception e) {}      l.fatal("This is the error message..");    System.out.println("Your logic executed successfully....");    }  } |

Once we run this client program, my.txt will contains….

## my.txt

FATAL – This is the error message..

## Explanation

* First step is to create one Logger class object [ see line number 9 ]
* Second step is to create Layout object  [ see line number 13 ]
* Once Layout is ready, our next step is to create Appender [ see line number 18 ]
* In appender i have passed 3 parameters like.. first parameter is object of layout because, appender will write the error messages based on the layout we selected, then 2nd parameter is file name with extension [ in this file only appender will writes the messages ], then 3rd parameter is by default true, means appender will appends the error messages, if we give false then appender will clears the previous data in my.txt file and write newly

Hey see, i have used FileAppender, but if i would like to change my appender choice to ConsoleAppender, then again we must open this java file then modifications and recompile bla bla…, so to avoid this we can use one .properties file, will see this in the next session.

# How To Create Log4j.properties File

In previous program, i have used FileAppender.  But if i would like to change my appender to JDBCAppender, i have to open my java file and do the modifications and need to recompile.  We can avoid this by  writing one .properties file.

By default the file name would be log4j.properties. This properties file stores data in the form of key, values pairs, in this file keys are fixed but values are our own.  We can include all the log4j related properties into this file.

## log4j.properties

log4j.rootLogger=DEBUG,CONSOLE,LOGFILE

log4j.appender.CONSOLE=  
log4j.appender.CONSOLE.layout=  
log4j.appender.CONSOLE.layout.ConversionPattern=

log4j.appender.LOGFILE=  
log4j.appender.LOGFILE.File=  
log4j.appender.LOGFILE.MaxFileSize=  
log4j.appender.LOGFILE.layout=  
log4j.appender.LOGFILE.layout.ConversionPattern=

will see one example.

# Example On Log4j.properties File With FileAppender & SimpleLayout

Let us see how to use log4j.properties file

## Files Required

* Client.java
* log4j.properties
* my.txt [ We will let the appender to write into this file ]

## Directory Structure

Client.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | import org.apache.log4j.Logger;    public class Client {      static Logger l = Logger.getLogger(Client.class.getName());      public static void main(String[] args) {          l.debug("This is debug message");        l.info("This is info message");        l.warn("This is warn message");        l.fatal("This is fatal message");        l.error("This is error message");          System.out.println("Your logic executed successfully....");      }  } |

Once we run this client program, my.txt will contains….

## log4j.properties

log4j.rootLogger = DEBUG,abc  
log4j.appender.abc = org.apache.log4j.FileAppender  
log4j.appender.abc.file = my.txt  
log4j.appender.abc.layout = org.apache.log4j.SimpleLayout

## my.txt

DEBUG – This is debug message  
INFO – This is info message  
WARN – This is warn message  
FATAL – This is fatal message  
ERROR – This is error message

## Execution Flow

* Run Client.java
* Log4j environment created, at line number 5
* As our default properties file name is **log4j**.properties, we no need to import properties file explicitly into Client.java, by default our java class will verify for the properties file named log4j.properties.  If we give the name other than log4j to the properties we have to import manually into our java class [ will see this later, like how to manually ]
* So once Logger object created at line number 5, our class will be able to know about the content in log4j.properties
* In log4j.properties the content always will be in key,value pairs only

## Explanation

* If we use .properties file, we no need to import any related classes into our java class
* log4j.rootLogger = DEBUG,abc  — > Here DEBUG means we are specifying the level from where log4j methods execution must start,  see my.txt file it showing all messages right.  But if we wrote log4j.rootLogger = WARN,abc then it will prints the messages in l.warn(), l.error(), l.fatal() and ignores l.debug(), l.info()
* I have used FileAppender as my Appender, so if we want to change my appender to ConsoleAppender, i will open log4j.properties file and do the modifications,  so no need to touch our java classes, this is the main advantage of .properties file

So just change layout into HTMLLayout and check the output

# Log4j Example On Using FileAppender And ConsoleAppender Simultaneously

Let us see how to use FileAppender and ConsoleAppender at a time.

## Files Required

* Client.java
* log4j.properties
* my.txt [ We will let the appender to write into this file ]

## Directory Structure

Client.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import org.apache.log4j.Logger;    public class Client {      static Logger l = Logger.getLogger(Client.class.getName());      public static void main(String[] args) {            l.debug("This is debug message");        l.info("This is info message");        l.warn("This is warn message");        l.fatal("This is fatal message");        l.error("This is error message");          System.out.println("Your logic executed successfully....");      }  } |

Once we run this client program, my.txt will contains….

## log4j.properties

log4j.rootLogger=DEBUG,CONSOLE,LOGFILE

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  
log4j.appender.CONSOLE.layout.ConversionPattern=%-4r [%t] %-5p %c %x – %m%n

log4j.appender.LOGFILE=org.apache.log4j.RollingFileAppender  
log4j.appender.LOGFILE.File=my.txt  
log4j.appender.LOGFILE.MaxFileSize=1kb  
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout  
log4j.appender.LOGFILE.layout.ConversionPattern=[%t] %-5p %c %d{dd/MM/yyyy HH:mm:ss} – %m%n

## my.txt

[main] DEBUG Client 27/02/2012 21:39:15 – This is debug message  
[main] INFO  Client 27/02/2012 21:39:15 – This is info message  
[main] WARN  Client 27/02/2012 21:39:15 – This is warn message  
[main] FATAL Client 27/02/2012 21:39:15 – This is fatal message  
[main] ERROR Client 27/02/2012 21:39:15 – This is error message