

**In Log4J 3 important objects are there**

(a) Logger object

**(b) Appender Object**

(c) Layout object

**Logger object**

=>Appender and Layout Objs must be linked with Logger obj directly or indirectly

=> Enables the logging on given class

=> allows to generate log message having different categories

=> Allows to specify logger level towards retrieving the log messages

**Appender object**

eg:: `Logger logger= Logger.getLogger(SelectTest.class);`

=> Will be linked with Logger object to specify the destination place to write/record log messages eg::

`FileAppender, ConsoleAppender, Rolling FileAppender, DailyRollingFileAppender, JdbcAppender, IMapAppender` and etc..

(best)

(assume max size is 30 MB)

`info.log.1`

`info.log. 2`

`info.log.3`

**Rolling FileAppender**

`info.log`

The class on which we want to enable the Logging

=>IMapAppender writes the log messages to email accounts as email messages =>JdbcAppender writes the log messages to DB s/w

(30MB) (30MB)

(30MB)

**DailyRollingFileAppender**

`info.log_22-04-2025`

(The log files will be created on hourly/daily/monthly/ /.../... in other ways) `info.log_21-04-2025`

`info.log_20-04-2025`

`info.log_19-04-202:5`

J

`localhost.2023-06-27`

`localhost.2023-06-28`

`localhost.2023-06-29`

`localhost.2023-07-03`

localhost.2023-07-04

localhost.2023-08-09

localhost.2023-09-09

localhost.2023-09-19 localhost.2023-09-21

## Layout object

=====

=>This object specifies the format to be used while writing the log messages specifies

=> This object the content and the order content that should come in the log messages

eg:: SimpleLayout, HTMLLayout, XMLLayout, PatternLayout

## Application

--> we can customize the format and content of the log messages

Layout objec

log messages

Logger object

Appender obj

Log destinations(file/console/...)

...

log messages

----

The log messages given to Logger obj will be written to destination Log files through appender obj according the format specified by the Layout object

## SLF4J

(Simple Logging Façade for Java)

on

=>To work with different logging tools, we need to use different apis.. So moving from 1 logging tool to another logging tool becomes very difficult to overcome this problem SLF4J is given which provides abstraction multiple logging tools or api and provides unified api to work any logging tool/api

Log4J (TV remote)

SLF4

(universal remote)

logback apache common-logging (AC remote)

(DTH Remote)

High level architecture of SLF4J

SLF4J unbound application

SLF4J bound to logback-classic

application

SLF4J bound to

log4j (best)

application

**SLF4J bound to java.util.logging**

application

SLF4J bound to simple application

SLF4J bound to no-operation

application

*SLF4J API slf4j-api.jar*

*SLF4J API slf4j-api.jar*

*SLF4J API slf4j-api.jar*

*SLF4J API slf4j-api.jar*

*SLF4J API slf4j-api.jar*

*SLF4J API slf4j-api.jar*

**/dev/null**

Underlying logging

Adaptation layer

Adaptation layer

Underlying

framework

*logback-classic.jar logback-core.jar*

*bridge jar file slf4j-log4j12.jar*

*bridge jar file slf4j-jdk14.jar*

Underlying logging framework

A

A invoking

software located

in B

**B**

*x.jar*

artifact available in classpath

x.jar SLF4J binding artifact available in classpath

Underlying logging framework

Underlying logging framework

*log4j.jar*

*JVM runtime*

non-native implementation

abstract logging api

native implementation

of slf4j-api (Given by SLESJ itself)

### adaptation layer (Bridge Layer)

of slf4j-api

Binding with a logging framework at deployment time

*logging framework slf4j-simple.jar*

### (slf4j itself providing the implementation)

*slf4j-nop.jar*

/dev/null

- SLF4J NOP binding is essentially a dummy or no-op implementation - meaning when you use this binding, all log calls will effectively do nothing. No logs will be written anywhere.

SLF4J Simple is a very lightweight logging backend that comes with SLF4J. Unlike NOP (which does

(Given by third party APIs) nothing), Simple actually prints log messages to the console (stdout).

It's useful when you want basic logging without adding bigger frameworks like Logback or Log4j.

As mentioned previously, SLF4J supports various logging frameworks. The SLF4J distribution ships with several jar files referred to as "SLF4J bindings", with each binding corresponding to a supported framework.

*slf4j-log4j12-1.7.28.jar*

Binding for log4j version 1.2, a widely used logging framework. You also need to place log4j.jar on your class path.

*slf4j-jdk14-1.7.28.jar*

Binding for java.util.logging, also referred to as JDK 1.4 logging

*slf4j-nop-1.7.28.jar*

Binding for NOP, silently discarding all logging.

*slf4j-simple-1.7.28.jar*

### (kills all the log messages)

Binding for Simple implementation, which outputs all events to System.err. Only messages of level INFO and higher are printed. This binding may be useful in the context of small applications.

*slf4j-jcl-1.7.28.jar*

Binding for Jakarta Commons Logging. This binding will delegate all SLF4J logging to JCL.

*logback-classic-1.2.3.jar (requires logback-core-1.2.3.jar)*

**NATIVE IMPLEMENTATION** There are also SLF4J bindings external to the SLF4J project, e.g. logback which implements SLF4J natively.

Logback's `ch.qos.logback.classic.Logger` class is a direct implementation of SLF4J's `org.slf4j.Logger` interface.

if no logging api/tool is

**specified for slf4j then internally uses logback by default**

**we can give instructions to slf4j and slf4j integrated with**

**log4j either using properties file(best) or using**

**xml file (not so recommended)**

**=> Log4j Logger object is useful for**

a) To enable logging on particular class

b) To categorize log messages

c) To filter log messages

=> Log4j Appender object is useful to write/record the log messages to different destinations

=> Log4j Layout object is useful to format the log messages

The logger levels of SLF4J are

debug<info<trace<warn<error

(no fatal here)

=>for user activity related event handling based code execution will be logged with the support of "trace" log message (Auditing activities) Button is clicked --> actionPerformed(-) method executed --> this can be logged through "trace" level

=> The code execution related to user signedIn and user signed out can be kept tracked /logged using trace logger level

Logger class that is given in SLF4J and in other logging apis is singleton java class i.e it allows us to create single object per JVM singleton ==> single (one) ton(object), So singleton means single object

Procedure to add SLF4J with log4j 1.x support to Java App for logging

=====

step1) Add the following jar files to the CLASSPATH or build path

> 010 slf4j-api-1.7.30.jar - C:\Us

>log4j-1.2.17 (1).jar - C:\Us

> l slf4j-log4j12-1.7.30.jar - C

> ojdbc6.jar

we can add in pom.xml

as maven dependencies

<https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12/1.7.31>

<https://mvnrepository.com/artifact/log4j/log4j/1.2.17>

<https://mvnrepository.com/artifact/org.slf4j/slf4j-api/1.7.31>

step2) place log4j.properties file in "src" folder..

log4j.properties

=====

(src/main/java folder)

#For HTMLLaout and FileAppender

#specify Logger level to retrieve the log messages

log4j.rootLogger=DEBUG,R

#specify appender

log4j.appender.R=org.apache.log4j.FileAppender

#Specify file name

log4j.appender.R.File=info.html

**#Disabling append mode on file**

**log4j.appender.R.append=true**

**#sepcify layaout**

**log4j.appender.R.layout=org.apache.log4j.HTMLLayout**

**step3) Develop any Java app having SLF4J based log messages**

**//SelectTest2.java**

**package com.nt.jdbc;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**import org.slf4j.Logger;**

**ojdbc8.jar**

**SLF4JWithLog41.xProj**

**✓ src**

**✓ com.nt.jdbc**

**=>In any Logging api or Logging framework based Logging activities we need to work with Logger object**

**> Select Test2.java**

**log4j.properties**

**object**

**This Logger can be used in multiple ways**

**a) To enable Logging on certain class**

**b) To write the log messages having different categories /priorities**

**> JRE System Library [JavaSE-15] Referenced Libraries >log4j-1.2.17 (2).jar - C:\Users\Nareshit\Do**

**>slf4j-log4j12-1.7.31.jar - C:\Users\Nareshit >slf4j-api-1.7.31.jar - C:\Users\Nareshit\Do > I ojdbc6.jar - C:\oraclexe\app\oracle\produ info.html**

**logger.info(".).**

**logger.debug(".).**

**.); .")**

**logger.trace(".. .....");**

**logger.error(".....") and etc..**

**In pom.xml (the dependencies are)**

**<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 --> <dependency>**

**<groupId>org.slf4j</groupId>**

**<artifactId>slf4j-log4j12</artifactId>**

```

<version>1.7.30</version>
</dependency>
<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.oracle.jdbc/ojdbc8 --> <dependency>
<groupId>com.oracle.jdbc</groupId>
<artifactId>ojdbc8</artifactId>
<version>19.3.0.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>1.7.30</version>
</dependency>
import org.slf4j.LoggerFactory;
public class SelectTest2
{
It is built-in property
of type java.lang.Class
private static Logger logger=LoggerFactory.getLogger(SelectTest2.class);
public static void main(String args[]){
logger.debug("SelectTest:: start of main(-) method");
Current Class name
giving the object
of the java.lang.Class holding the class name
given
Connection con=null;
Statement st=null;
ResultSet rs=null;
try {
//Load jdbc driver class
//establish the connection (Road)

```

```

Class.forName("oracle.jdbc.driver.Oracle Driver");
logger.debug("com.nt.jdbc.SelectTest:: JDBC driver driver class loaded");
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system", "manager");
logger.info("com.nt.jdbc.SelectTest:: Connection is established with DB s/w"); //create JDBC Statement
object (vechicle)
if(con!=null) {
st=con.createStatement();
logger.debug("com.nt.jdbc.SelectTest: JDBC Statement object is created");
}
//Send and execute SQL SELECT Query in Db s/w and get JDBC ResultSet object
if(st!=null) {
rs= st.executeQuery("SELECT * FROM STUDENT");
logger.debug("com.nt.jdbc.SelectTest: SQL query is send to Db s/w for execution and ResultSet obj is
generated");
}
if(rs!=null) {
//process the ResultSet object
while(rs.next()!=false){ // while(rs.next()==true)
// System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+ rs.getString(3)+" "+rs.getFloat(4));
//System.out.println(rs.getInt("SNO")+" "+rs.getString("SNAME")+" "+ rs.getString("SADD")+"
"+rs.getFloat("AVG"));
//System.out.println(rs.getString("SNO")+" "+rs.getString("SNAME")+" "+rs.getString("SADD")+"
"+rs.getString("AVG"));
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+ rs.getString(3)+" "+rs.getString(4));
} //while
logger.warn("com.nt.jdbc.SelectTest:: The records are ResultSet obj are retrived using getStrting(-) for all
cols ..change them accordingly"); logger.debug("com.nt.jdbc.SelectTest::ResultSet obj is processed");
} //if
} //try
catch(SQLException se) {
}
se.printStackTrace();
logger.error("com.nt.jdbc.SelectTest:: known DB Problem ::"+se.getMessage()+" SQL error
code"+se.getErrorCode());
catch (Exception e) {
e.printStackTrace();
logger.error("com.nt.jdbc.SelectTest:: unknown Problem "+e.getMessage());
}
}

```



```

finally {
logger.debug("com.nt.jdbc.SelectTest: Closing JDBC objs");
//close jdbc objs
try {
if(rs!=null)
rs.close();
logger.debug("com.nt.jdbc.SelectTest: ResultSet obj is closed");
}
catch(SQLException se) {
se.printStackTrace();
logger.error("com.nt.jdbc.SelectTest: Problem in closing ResultSet obj "+se.getMessage());
}
try {
if(st!=null)
st.close();
logger.debug("com.nt.jdbc.SelectTest: Statement obj is closed");
}
catch(SQLException se) {
se.printStackTrace();
logger.error("com.nt.jdbc.SelectTest: Problem in closing Statement obj "+se.getMessage());
}
try {
if(con!=null)
con.close();
logger.debug("com.nt.jdbc.SelectTest: Connection obj is closed");
}
catch(SQLException se) {
se.printStackTrace();
logger.error("com.nt.jdbc.SelectTest: Problem in closing Connection obj "+se.getMessage());
}
}finally
logger.debug("com.nt.jdbc.SelectTest: end of main(-) method");
}main
}///class

```

note:: Here SLFJ generated the log messages by using log4j setup internally based on the instructions collected from the log4j.properties file.

uses

**note: spring boot internally slf4j with log4j to generate log messages.. we can control these log messages through application.properties file..**

**=>Built-in reference variables in Java are "this","super".**

**=>Built-in threads in java are "main","gc" (garbage collector) =>Built-in streams in java are "System.in","System.out","System.err"**

**=>Built-in properties in java are "class","length"**

**length (useful to get the length of the array)**

**int a[]=new int[]{10,20,30};**

**int size=a.length; //gives the size of the array**

**class (gives the object of java.lang.Class)**

**Class c1=Test.class; Class c2=java.util.Date.class; obj of java.lang.Class**

**Test class info**

**--> class name,**

**-->super class name**

**"class" is the static property that will be**

**added by the java compiler in every java class automatically and dynamically.. this property gives object of java.lang.Class having the meta data of the given class object of java.lang.Class java.util.Date class info**

**-->class name -->super class name**

**|-->method details**

**MetaData about Test class**

**|--->constructor details**

**|--->**

**|--> ....**

**In a running java app,**

**=> we use numeric variables (int,long,short and etc..) to hold the numeric values**

**=> we use floating variables(float,double) to hold floating point values**

**=> we use String variables to hold text content**

**=> Similarly we can use the object of java.lang.Class to hold java class/interface/Enum/annotation/.. and its metadata**

**In LoggerFactory.getLogger(-) method,we are passing SelectTest.class that means**

**we are passing the object of java.lang.Class having SelectTest class and its meta data..**

**Project using maven**

**SL4FJProj01-SL4FJWithLog4J**

**src/main/java**

**com.nt.service**

**> SelectTest.java**

**log4j.properties**

**> src/test/java**

> JRE System Library [JavaSE-17]

Maven Dependencies

> junit-4.11.jar - C:\Users\NATARAJ\.m2\repository\junit\junit\4.1

> hamcrest-core-1.3.jar - C:\Users\NATARAJ\.m2\repository\org\

> slf4j-log4j12-1.7.30.jar - C:\Users\NATARAJ\.m2\repository\org\

> log4j-1.2.17.jar - C:\Users\NATARAJ\.m2\repository\log4j\log4j

> ojdbc8-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com\or

>

>

ucp-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com\orac

oraclepki-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com\

> osdt\_cert-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com\

> osdt\_core-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com'

> simplefan-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com'

> ons-19.3.0.0.jar - C:\Users\NATARAJ\.m2\repository\com\oracle

> slf4j-api-1.7.30.jar - C:\Users\NATARAJ\.m2\repository\org\slf4j

> src

> target

info\_log.html

pom.xml

**sample info\_log.html**

Log session start time Sat Sep 23 10:10:07 IST 2023

**Category**

com.nt.service.SelectTest com.nt.service.SelectTest com.nt.service.SelectTest

**Time**

**Thread**

**Level**

0

main

DEBUG

2

main

DEBUG

160

main

INFO

440

main

INFO

509

main

INFO

com.nt.service.SelectTest

540

main

INFO

com.nt.service.SelectTest

543

main

INFO

com.nt.service.SelectTest

543

main

DEBUG

com.nt.service.SelectTest

com.nt.service.SelectTest

**Message**

start of main(-) method

start of try block

JDBC driver class is loaded Connection is established PreparedStatement obj is created

JDBC ResultSet object is created JDBC ResultSEt object processed

end of try block

**log4j.rootLogger=ALL,R**

**log4j.appender.R=org.apache.log4j.FileAppender**

**Entries in log4.properties (For writing the log messages to XML file)**

**# To work with XMLLayout and FileAppender**

**#Specify the logger level to retrieve the log messages**

**log4j.appender.R.File=info\_log1.xml**

Log output

Conversion

Performance

Character

**log4j.appender.R.append=true**

Priority (level)

P

Fast

**log4j.appender.R.layout=org.apache.log4j.xml.XMLLayout**

New line separator Thread name

n

Fast

t

Fast

Time elapsed (milliseconds)

Γ

Fast

Thread's nested diagnostic context

X

Fast

**log4j.properties (For ConsoleAppender and PatternLayout)**

Thread's mapped diagnostic context

**x**

Fast

Percent sign

%%

Fast

**# To work with PatternLayout and ConsoleAppender**

Category name (or logger name)

c

Fast

**log4j.rootLogger=ALL,R**

Log message

m

Fast

**log4j.appender.R=org.apache.log4j.ConsoleAppender**

Fully qualified class name

C

Slow

Date and time

d

*Slow if using JDK's formatters.*

**log4j.appender.R.Target=System.out**

**log4j.appender.R.layout=org.apache.log4j.PatternLayout**

d{format}

*Fast if using log4j's formatters.*

**log4j.appender.R.layout.Conversion Pattern=%p-%t-%r-%c-%m-%d-%n**

File name of Java class

F

Extremely slow

Location (class, method and line number)

Extremely slow

**note:: Pattern Layout allows to specify the order of content that we want to see in log message.**

Line number only Method name

L

Extremely slow

M

Extremely slow

**popular log4j layout is :: PatternLayout**

**log4j.properties (For Working with RollingFileAppender and PatternLayout)**

**RollingFileAppender** allows us to specify max size for the log file .. once that max file size is reached, the backup files will be recreated..

**# To work with PatternLayout and RollingFileAppender**

**log4j.rootLogger=ALL,R**

**log4j.appender.R=org.apache.log4j.RollingFileAppender**

**log4j.appender.R.File=roll\_info\_log.txt**

**log4j.appender.R.Append=true**

**note:: Any Layout can be used with any Appender by applying basic common sense**

**roll\_info\_log.txt**

**roll\_info\_log.txt .1**

**roll\_info\_log.txt.2**

... ..

**log4j.appender.R.Immediate Flush=true**

**log4j.appender.R.MaxFileSize=5KB**

**log4j.appender.R.MaxBackupIndex=3**

**log4j.appender.R.layout=org.apache.log4j.PatternLayout**

**log4j.appender.R.layout.ConversionPattern=%p-%-%%-%%m-%d-%n**

current file

...

...  
... ..

5KB

roll\_info\_log.txt .3

.....

...

5KB

5KB

Example on DailyRollingFileAppender and PatternLayout

=====

##### Using DailyRollingFileAppender and PatternLayout #####

### specify the Logger Level to retrieve the Log message

log4j.rootLogger=DEBUG,R

###specify the appender name

log4j.appender.R=org.apache.log4j.Daily RollingFileAppender

###specify the file name

log4j.appender.R.File=d\_roll\_log\_info.txt

# enable the append mode

log4j.appender.R.File.Append=true

#Date pattern

log4j.appender.R.Date Pattern='yyyy-MM-dd-HH-mm

#enable the immediateFlush

log4j.appender.R.immediate Flush=true

###Specify the Layout

log4j.appender.R.layout=org.apache.log4j.PatternLayout

###Specify the Conversion Pattern

log4j.appender.R.layout.Conversion Pattern=%p - %t - %c -% -%d-%m%n

-yyyy-MM -->gives every month one log file

- "yyyy-MM-dd ---->gives every day one log file

· "yyyy-MM-dd-HH -->gives every hour one log file

- "yyyy-MM-dd-HH-mm

----> gives every minute one log file

--.'yyyy-ww---> gives every week one log file

-- .'yyyy-MM-dd-a----> gives log file every midday

and midnight

In Real Projects, we generally deal with two log files

a) Common Log file1 ---- To record all the log messages of all the levels (Logger Level is:: DEBUG)

**b) Error Log file2 ----->To record only Exception/Error related log messages (Logger Level is :: ERROR)**

**Log4j.properties (To deal with multiple log files)**

**#Working with multiple log files and Loggers in one Application**

log4j.rootLogger=ALL,R

log4j.logger.com.nt.main

**# For All log messages**

**=ERROR, E**

**log4j.appender.R=org.apache.log4j.FileAppender**

log4j.appender.R.File=droll\_info\_log.txt

log4j.appender.R.Append=true

log4j.appender.R.Immediate Flush=true

**log4j.appender.R.layout=org.apache.log4j.PatternLayout**

**R ---> For all log messages to a common Log file (Logger Level is DEBUG)**

**E ---> For Error log messages to a Error Log file (Logger Level is ERROR)**

log4j.appender.R.layout.Conversion Pattern=%p- %t-%r-%c-%m-%d-% -%n

**# For Exception log messages**

**log4j.appender.E=org.apache.log4j.DailyRollingFileAppender**

**log4j.appender.E.File=droll\_info\_log\_errors.txt**

log4j.appender.E.DatePattern='.' yyyy-MM-dd-HH-mm

log4j.appender.E.Append=true

log4j.appender.E.Immediate Flush=true

**log4j.appender.E.layout=org.apache.log4j.PatternLayout**

log4j.appender.E.layout.Conversion Pattern-%-%%-%%-%%m-%d-% -%n

**FAQS**

**Q) What is Logging?**

**Q) what is the difference b/w logging and auditing?**

**Q) What are problems with System.out.println(-) based logging operations?**

**Q) List out various Logging APIs/Tools/Frameworks?**

**Q) How SLF4J is different from regular Log4j and other Logging tools**

**Q) What is the default Logging API that is linked with SLF4J**

**Logger Levels and their usecases?**

**Q) Explain different**

**Q)**

**Explain Different Appenders**

**Q) Explain different Layouts**

**Q) Explain Logger Object**

**Q) Advantages**



of doing Logging using various logging tools over System.out.println(-)?

Q) What is the need of two different logging files in Realtime Projects Development?

Q) What is difference b/w Rolling FileAppender and DailyRolling FileAppender?

Q) List out various details about PatternLayout

Q) Explain the format specifiers of the PatternLayout

Q) While integrating log4j with SLF4J what is file that we need to use to provide configurations?

(log4j.properties or log4j.xml)

Q) What is the popular Appender and Layout that will be used in Logging operations?

(DailyRollingFileAppender and PatternLayout)

Q) What is Html Layout and PatternLayout?