Core Java 8 and Development Tools

Lesson 19 : Logging with Log4J

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to

- Understand the necessity of Logging
- Work with log4j components
- Configure java application for logging

19.1: Introduction
Overview of Logging

Logging is writing the state of a program at various stages of its execution to some repository such as a log file.

By logging, simple yet explanatory statements can be sent to text file, console, or any other repository.

Using logging, a reliable monitoring and debugging solution can be achieved.

19.1: Introduction
Logging Requirements

Logging is used due to the following reasons:

- It can be used for debugging.
- It is cost effective than including some debug flag.
- There is no need to recompile the program to enable debugging.
- It does not leave your code messy.
- Priority levels can be set.
- Log statements can be appended to various destinations such as file, console, socket, database, and so on.
- Logs are needed to quickly target an issue that occurred in service.

19.2: Log4J Concepts
Concept of Log4J

Log4j is an open source logging API for Java.

- It handles inserting log statements in application code, and manages them externally without touching application code, by using external configuration files.
- It categorizes log statements according to user-specified criteria and assigns different priority levels to these log statements.
- It lets users choose from several destinations for log statements, such as console, file, database, SMTP servers, GUI components.
- It facilitates creation of customized formats for log output and provides default formats.

19.2: Log4J Concepts Components

Log4j comprises of three main components:

- Logger
- Appender
- Layout

Users can extend these basic classes to create their own loggers, appenders, and layouts.

19.2: Log4J Concepts
Logger

The component logger accepts log requests generated by log statements.

Logger class provides a static method getLogger(name).

- This method:
  - Retrieves an existing logger object by the given name (or)
  - Creates a new logger of given name if none exists.
- It then sends their output to appropriate destination called appenders.

19.2: Log4J Concepts
Logger

The logger object is then used to ….

- set properties of logger component
- invoke methods which generate log requests, namely:
  - debug(), info(), warn(), error(), fatal(), and log()

Each class in the Java application being logged can have an individual logger assigned to it or share a common logger with other classes.

Any number of loggers can be created for the application to suit specific logging needs.

19.2: Log4J Concepts
Logger

Log4j provides a default root logger that all user-defined loggers inherit from.

- Root logger is at the top of the logger hierarchy of all logger objects that are created.
- If an application class does not have a logger assigned to it, it can still be logged using the root logger.
- Root logger always exists and cannot be retrieved by name.

19.2: Log4J Concepts
Ways to create a Logger

Retrieve the root logger:

```
Logger logger = Logger.getRootLogger();
```

Create a new logger:

```
Logger logger = Logger.getLogger("MyLogger");
```

Instantiate a static logger globally, based on the name of the class:

```
static Logger logger = Logger.getLogger(test.class);
```

Set the level with:

```
logger.setLevel((Level)Level.WARN);
```

19.2: Log4J Concepts
Logger Priority Levels

Loggers can be assigned different levels of priorities.

Priority levels in ascending order of priority are as follows:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

19.2: Log4J Concepts
Logger Priority Levels

There are printing methods for each of these priority levels.

mylogger.info("logstatement1");

- generates log request of priority level INFO for logstatement1

19.2: Log4J Concepts
Logger Priority Levels

In addition, there are two special levels of logging available:

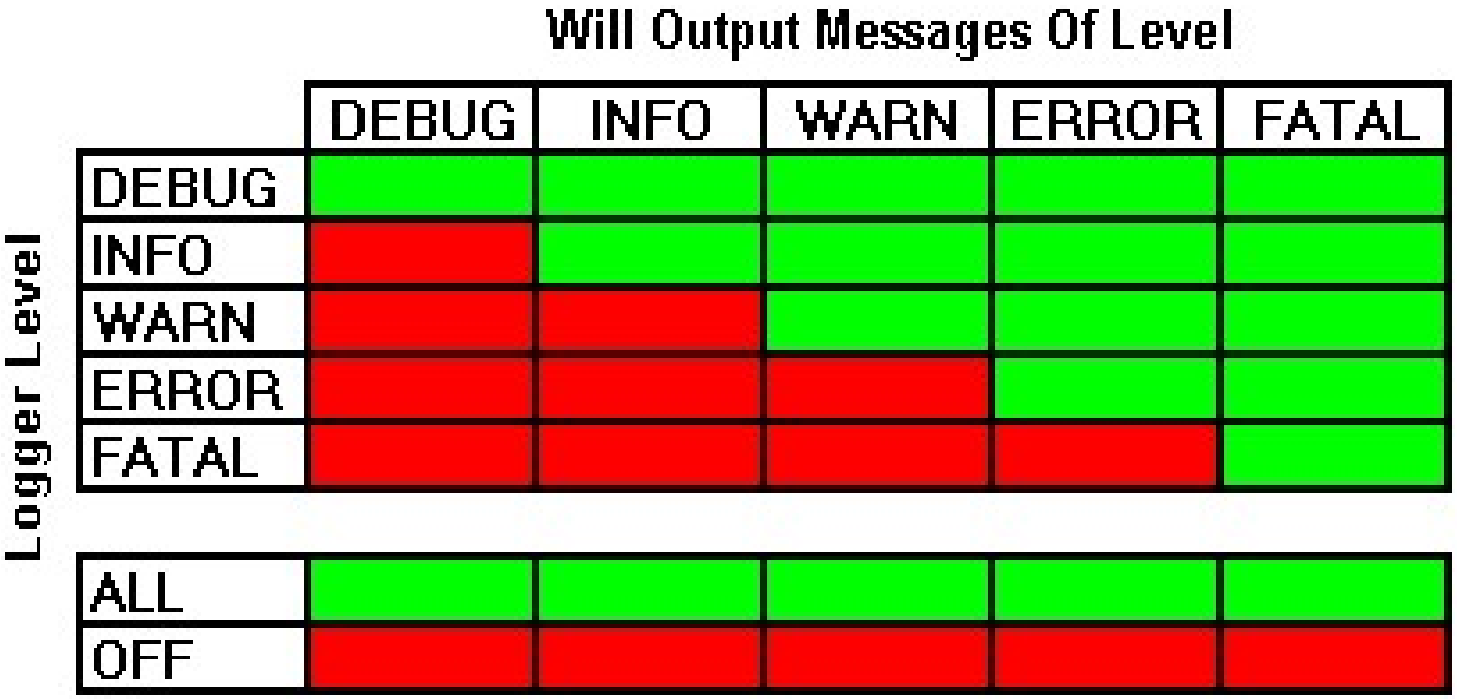
- ALL: It has lowest possible rank. It is intended to turn on all logging.
- OFF: It has highest possible rank. It is intended to turn off logging.

The behavior of loggers is hierarchical.

The root logger is assigned the default priority level DEBUG.

19.2: Log4J Concepts
Logger Priority Levels

- The behavior of loggers is hierarchical. The following table illustrates this situation.
  - Logger Output Hierarchy:



19.2: Log4J Concepts
Logger Priority Levels

A logger will only output messages that are of a level greater than or equal to it.

If the level of a logger is not set, it will inherit the level of the closest ancestor.

If a logger is created in the package com.foo.bar and no level is set for it, then it will inherit the level of the logger created in com.foo.

19.2: Log4J Concepts
Logger Priority Levels

If no logger was created in com.foo, then the logger created in com.foo.bar will inherit the level of the root logger.

The root logger is always instantiated and available. The root logger is assigned the level DEBUG.

19.2: Log4J Concepts
Examples on Priority Levels

```
/* Instantiate a logger named MyLogger */
   Logger mylogger = Logger.getLogger("MyLogger");
/* Set logger priority l*/
    mylogger.setLevel(Level.INFO);
/* statement logged, since INFO = INFO*/
    mylogger.info(" values ");
/* statement not logged, since DEBUG < INFO*/ mylogger.debug(“not logged");
/* statement logged, since   ERROR >INFO*/  mylogger.error(“logged");
```

19.2: Log4J Concepts
Appender

Appender component is interface to the destination of log statements, a repository where the log statements are written/recorded.

A logger receives log request from log statements being executed, enables appropriate ones, and sends their output to the appender(s) assigned to it.

The appender writes this output to repository associated with it.

- Examples: ConsoleAppender, FileAppender, WriterAppender, RollingFileAppender, DailyRollingFileAppender

19.2: Log4J Concepts
Layout

The Layout component defines the format in which the log statements are written to the log destination by “appender”.

Layout is used to specify the style and content of the log output to be recorded.

- This is accomplished by assigning a layout to the appender concerned.

Layout is an abstract class in log4j API.

- It can be extended to create user-defined layouts.

19.2: Log4J Concepts
Types of Layout

Let us discuss the different types of layouts:

- HTMLLayout: It formats the output as a HTML table.
- PatternLayout: It formats the output based on a conversion pattern specified. If none is specified, then it uses the default conversion pattern.
- SimpleLayout: It formats the output in a very simple manner, it prints the Level, then a dash "-", and then the log message.
- XMLLayout: It formats the output as a XML.

19.3: Installation of Log4J
Steps

Let us see the steps for installation of Log4J:

- Download log4j-1.2.4.jar from http://logging.apache.org/log4j
- Extract the log4j-1.2.4.jar at any desired location and include its absolute path in the application's CLASSPATH.
- Now, log4j API is accessible to user's application classes and can be used for logging.

Steps for Installation of Log4J

Notes are there.

Steps for Installation of Log4J

Notes are there.

19.4: Configuring Log4J Process

The log4j can be configured both programmatically and externally using special configuration files.

External configuration is most preferred.

- This is because to take effect it does not require change in application code, recompilation, or redeployment.

Configuration files can be XML files or Java property files that can be created and edited using any text editor or xml editor, respectively.

19.4: Configuring Log4J Using Property File

The root logger is assigned priority level DEBUG and an appender named myAppender:

- log4j.rootLogger=debug, myAppender

The appender's type specified as ConsoleAppender, that is logs output to console:

- log4j.appender.myAppender=org.apache.log4j.ConsoleAppender

The appender is assigned a layout SimpleLayout:

- log4j.appender.myAppender.layout=org.apache.log4j.SimpleLayout

19.4: Configuring Log4J
Using Configuration Property Files

Example 1:

```
package com.igate.sampleapp;
import org.apache.log4j.*;
public class MyClass
{     static Logger myLogger =
Logger.getLogger(MyClass.class.getName( ));
       //no constructor required
      public void do_something( int a, float b)
      {      myLogger.info("Logged since INFO=INFO"); ...
             myLogger.debug("not enabled, since DEBUG < INFO");
      ...                                                  contd.
```

19.4: Configuring Log4J
Using Configuration Property Files

Example 1 (contd.):

```
                    if (x == null)
                    {myLogger.error("enabled & logged ,since ERROR >
INFO");...}
     }
 }
```

- //Execute the file as
  // java -Dlog4j.configuration=config-sample.properties
  com.igate.sampleapp.MyClass

19.4: Configuring Log4J
Using Configuration Property Files

Example 2:

```
package com.igate.sampleapp;
import org.apache.log4j.*;
import org.apache.log4j.PropertyConfigurator;
import java.net.*;
public class MyClass
{     static Logger myLogger =
Logger.getLogger(MyClass.class.getName( ));
      public MyClass()
      {
                  PropertyConfigurator.configure("config-
sample.properties");
      }
                        contd.
```

19.4: Configuring Log4J
Using Configuration Property Files

Example 2 (contd.):

```
public void do_something( int a, float b)
{           myLogger.info("Logged since INFO=INFO"); ...
            myLogger.debug("not enabled, since DEBUG < INFO");
...
      if (x == null) {
          myLogger.error("enabled & logged ,since ERROR > INFO");...
       }
}
```

19.4: Configuring Log4J Demo

MyClass_Property.java

19.5: Pros and Cons
Log4j

Advantages:

- Log4j print methods do not incur heavy process overhead.
- External configuration makes management and modification of log statements simple and convenient.
- Priority level of log statements helps to weed out unwanted logging.

ShortComings:

- Appender additivity may result in log requests being sent to unwanted appenders.

Lab : Log4J

Lab 13: Log4J

Summary

In this lesson, you have learnt

- Log 4j and its components
- The method to log messages using Log4J API
- Configuring Log4j applications

Review Question

Question 1: Log4j is a product of ___.

- Option1: RedHat
- Option2: Sun
- Option3: Apache
- Option4: None of the above

Review Question

Question 2: Configuration of Log4j applications can be done through:

- Option1: Property files
- Option2: XML files
- Option 3: Both Option1 and Option2 are right
- Option4: None of the above is true

Review Question

Question 3: If the loglevel is set to INFO, will the following message be logged?
Message: logger.warn(“ From warning");

- Yes / No