Apache log4j Logger

Introduction

Logging within the context of program development constitutes inserting statements into the program that provide some kind of output information that is useful to the developer. Examples of logging are trace statements, dumping of structures and the familiar System.out.println or printf debug statements. log4j offers a hierarchical way to insert logging statements within a Java program. Multiple output formats and multiple levels of logging information are available.

By using a dedicated logging package, the overhead of maintaining thousands of System.out.println statements is alleviated as the logging may be controlled at runtime from configuration scripts. log4j maintains the log statements in the shipped code. By formalising the process of logging, some feel that one is encouraged to use logging more and with higher degree of usefulness.

Log4j Basic Concepts

1. public class Logger

   Logger is responsible for handling the majority of the log operation.
2. public interface Appender

   Appender is responsible for controlling the output of the log operation.
3. public abstract class Layout

   Layout is responsible for formatting the output for Appender.

Log Level

The logger is the core component of the logging process. In log4j, there are five normal Levels of logger available (not including custom Levels).

The log4j levels follow the following order.

- *DEBUG*
- *INFO*
- *WARN*
- *ERROR*
- *FATAL*

Example :

```
log4j.rootLogger=DEBUG, CA

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

```
package com.vaannila.helloworld;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
public class HelloWorld {

    static final Logger logger = Logger.getLogger(HelloWorld.class);

    public static void main(String[] args) {
        PropertyConfigurator.configure("log4j.properties");
        logger.debug("Sample debug message");
        logger.info("Sample info message");
        logger.warn("Sample warn message");
        logger.error("Sample error message");
        logger.fatal("Sample fatal message");
    }
}
```

When you execute the example the following output will be displayed on the console.

```
 [main] DEBUG com.vaannila.helloworld.HelloWorld  - Sample debug message
 [main] INFO  com.vaannila.helloworld.HelloWorld  - Sample info message
 [main] WARN  com.vaannila.helloworld.HelloWorld  - Sample warn message
 [main] ERROR com.vaannila.helloworld.HelloWorld  - Sample error message
 [main] FATAL com.vaannila.helloworld.HelloWorld  - Sample fatal message
```

If you set the rootLogger level to *WARN* then only the *WARN*, *ERROR* and *FATAL* level log messages will be displayed and the rest will be dropped.When you execute the example the following output will be displayed on the console.

```
[main] WARN  com.vaannila.helloworld.HelloWorld  - Sample warn message
[main] ERROR com.vaannila.helloworld.HelloWorld  - Sample error message
[main] FATAL com.vaannila.helloworld.HelloWorld  - Sample fatal message
```

```
log4j.rootLogger=DEBUG, CA

log4j.appender.CA=org.apache.log4j.ConsoleAppender

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

Level	Description
FATAL	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARN	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.

INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	Detailed information on the flow through the system. Expect these to be written to logs only.
TRACE	More detailed information. Expect these to be written to logs only. Was only added in version 1.2.12.

Configuration

There are two ways to configure log4j. One is with a properties file and the other is with an XML file. Configuring logging via a file has the advantage of turning logging on or off without modifying the application that uses log4j.

Example log4j configuration properties file

log4j.properties

```
log4j.rootLogger=DEBUG

# AdminFileAppender - used to log messages in the admin.log file.
log4j.appender.AdminFileAppender=org.apache.log4j.FileAppender
log4j.appender.AdminFileAppender.File= admin.log
log4j.appender.AdminFileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.AdminFileAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

# ReportFileAppender - used to log messages in the report.log file.
log4j.appender.ReportFileAppender=org.apache.log4j.FileAppender
log4j.appender.ReportFileAppender.File= report.log
log4j.appender.ReportFileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.ReportFileAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

log4j.logger.com.java.logger. SampleAdmin =,AdminFileAppender
log4j.logger.com.java.logger. SampleReport =,ReportFileAppender
```

Example log4j configuration XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
        <appender name="stdout" class="org.apache.log4j.ConsoleAppender">
                <layout class="org.apache.log4j.PatternLayout">
                        <param name="ConversionPattern" value="%d{ABSOLUTE} %5p
%c{1}:%L - %m%n"/>
                </layout>
        </appender>
        <root>
                <priority value="debug"></priority>
                <appender-ref ref="stdout"/>
        </root>
</log4j:configuration>
```

Tutorial (Apacahe Log4j)

If we use the apache log4j API, we have to class path log4j.jar file and configuration file. The following examples will be log information on the log file according to the above log4j.properties file configuration. So, we have to class path your destination log file(Eg : admin.log and report.log).

```
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class SampleAdmin {
   static Logger logger = Logger.getLogger(SampleAdmin.class);

   public static void main(String[] args) {
      logger.debug("Sample debug message");
      logger.info("Sample info message");
      logger.warn("Sample warn message");
      logger.error("Sample error message");
      logger.fatal("Sample fatal message");
   }
}
```

```
import org.apache.log4j.Logger;

public class SampleReport {

   static Logger logger = Logger.getLogger(SampleReport.class);

   public static void main(String[] args) {
      logger.debug("Sample debug message");
      logger.info("Sample info message");
      logger.warn("Sample warn message");
      logger.error("Sample error message");
      logger.fatal("Sample fatal message");
   }
}
```

HTML Layout

If we want to log message as html output format, we only need to add some of the configuration into the log4j configuration file.

```
Log4j.rootLogger=DEBUG
# AdminFileAppender - used to log messages in the admin.html file.
log4j.appender. AdminFileAppender =org.apache.log4j.FileAppender
log4j.appender. AdminFileAppender.File= admin.html
log4j.appender. AdminFileAppender.layout=org.apache.log4j.HTMLLayout
log4j.appender. AdminFileAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```