Log4J

Log4J Workshop Content

 Introduction to Logging

 What is Logging?

 Importance of Logging

 Available Logging Frameworks for JAVA

 Introduction to Log4J

 What is Log4J?

 Why we need to use Log4J?

 Log4J Features

 Log4J Advantages

 Log4J Setup for Java: Standalone and Web Applications

 Download Log4J

 log4J Jar file

 log4j.properties

 Log4J Development Approaches

 Programmatic

 Declarative

 Log4J Programmatic Implementation

 Logger

 BasicConfiguror, PropertyConfigurator

 Setting Logging Level

 Log4J Logging Levels

 Default Logging Level

 Available Logging Levels

DEBUG, INFO, WARN, ERROR and FATAL

 Log4J Levels Hierarchy

 Log4J Logger

 RootLogger

 Logger

 Relationship between RootLogger and Logger

 Log4J Declarative Implementation: log4j.properties Configuration

 Set Debug Level

 Set Appender

 Set PatternLayout

 Set ConversionPattern

 Log4J Appenders

 ConsoleAppender

 FileAppender

 RollingFileAppender

 AdminFileAppender

 ReportFileAppender

 Setting Single Appender

 Setting Multiple Appenders

 Log4J Implementation with xml file

 log4j.xml

 <log4j:configuration>

 Setting Appenders

 Setting Log Level

 Setting Conversion Pattern

 Conversion Pattern Syntax

 TTCC

 TTCCLayout

 Time Elapsed

 Thread Information

 Logging Priority

 Logging Category

 NDC - Nested Diagnostic Context

 Application Message

 Log4J Integration

 Java Standalone Project

 Servlets/JSP Project

 Log4J Issues

 Log4J Drawbacks

 Alternative to Log4J

 Introduction to SLF4J

 Advantages of SLF4J

 Migration of Logging Framework

 Log4J to SLF4J

Log4J

Introduction to Logging:

What is Logging?

Logging is a technique or process to record the development activities to a console

or a log file.

Several Logging frameworks simplify and standardize the process of logging for the

Java platform.

Why we need logging in an application?

 To understand the flow of the application logic

 To find out root cause of an issue

 To track transactions (In Banking Domain) permanently by storing log into a

Data base.

Advantages of Logging:

 Easy to debug application

 Easy to find out root cause of the problem

 Easy to find out flow of the logic

Drawbacks of Logging:

 Logging severely affects the performance of the application

 It consumes some memory at server side to store data.

In Realtime, most of the developers uses different logging techniques during

development to understand his/her code execution process and also to find out root

cause of the problem.

As a less experience developer or not working people, you may like to use

System.out.println(logMessage)

Statement to log your statements to a console(Command prompt).

But it is NOT recommended to use System.out.println(logMessage)

in real time. It has lot of advantages.

What is the simple logging technique available for a Java Program?

Using System.out.println(logMessage)

What is the major drawback of System.out.println(); logging technique?

When we want to deliver code from development to next phase or into production,

we have to remove or comment all SOP statements one by one manually.

 It is very time consuming process

 It kills our development time

 It increases development time and cost

That’s why most of the people uses one of the available logging frameworks in their

application.

Popular Logging Frameworks for Java Applications:

S.No. Logging Frameworks for JAVA

1 Java Logging API

2 Log4J

3 Apache Commons Logging

4 SLF4J

SLF4J stands for Simple Logging Facade for Java

Java Logging API

Sun MircoSystmes (Oracle Corporation) has introduced one logging framework as part of JSE(Java

Standard Edition) API under java.util.logging package.

Sample API Classes

java.util.logging.FileHandler

java.util.logging.Level

java.util.logging.LogManager

java.util.logging.Logger

java.util.logging.XMLFormatter

Sample Java Logging Program:

import java.util.logging.Level;

import java.util.logging.Logger;

public class SampleProgram

{

public static void main(String[] args)

{

Logger log = Logger.getLogger("Some Logging");

log.info("Its an INFO Message");

log.log(Level.SEVERE, "Its an INFO Message");

log.info("Its an INFO Message");

}

}

Output:

Jun 2, 2011 3:00:30 PM SampleProgram main

INFO: Its an INFO Message

Jun 2, 2011 3:00:30 PM SampleProgram main

SEVERE: Its an SEVERE Message

Jun 2, 2011 3:00:30 PM SampleProgram main

INFO: Its an INFO Message

JSE Logging API contains some drawbacks

 Does NOT contain meaningful logging levels

 Performance issues

 Less flexible to use

NOTE:-

Unlike other frameworks - Log4J, Java Logging API does NOT flexible API to use

different configuration files like properties file, xml file etc.

The levels in descending order are:

 SEVERE (highest value)

 WARNING

 INFO

 CONFIG

 FINE

 FINER

 FINEST (lowest value)

Log4J solves most of these problems and provides many advantages.

LOG4J Framework

What is Log4J?

Log4J stands for Logging Framework for Java

It is an open source logging framework from Apache Software Foundation for Java

Applications

Log4J is the most popular logging framework for Java Applications (for both

Standalone Java Applications and Web applications).

Log4j is JDK 1.1.x compatible.

Log4J Framework Advantages:

 Contains meaningful logging levels

 Resolves some Performance issues

 Easy and flexible to use

 Supports both properties file configurations and XML configurations

Drawbacks of Log4J Framework:

 Reduces some application performance

 Tightly coupled

 Bit tough to migrate Log4J framework to other framework

Log4J Framework Features:

 log4j is optimized for speed.

 log4j is based on a named logger hierarchy.

 log4j is fail-stop but not reliable.

 log4j is thread-safe.

 log4j is not restricted to a predefined set of facilities.

 Logging behavior can be set at runtime using a configuration file.

Configuration files can be property files or in XML format.

 log4j is designed to handle Java Exceptions from the start.

 log4j can direct its output to a file, the console, an java.io.OutputStream,

java.io.Writer, a remote server using TCP, a remote Unix Syslog daemon, to

a remote listener using JMS, to the NT EventLog or even send e-mail.

 log4j uses 5 levels, namely DEBUG, INFO, WARN, ERROR and FATAL.

 The format of the log output can be easily changed by extending the Layout

class.

 The target of the log output as well as the writing strategy can be altered by

implementations of the Appender interface.

 log4j supports multiple output appenders per logger.

 log4j supports internationalization.

NOTE:-

To overcome this tightly couple problem, Spring Framework has introduced AOP (Aspect Oriented

Programming). In AOP, we can do logging dynamically and declaratively at runtime.

That means Logging component won’t touch our application program.

Log4J supports the following logging levels

ALL

DEBUG

INFO

WARN

ERROR

FATAL

OFF

For normal log levels in Log4J Framework, the ascending log levels are

DEBUG

INFO

WARN

ERROR

FATAL

Log4J Logging levels

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.

WARNING

Use of deprecated APIs, poor use of API, near 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.

OFF

Switch off Logging completely Or

Turns Off all logging

ALL Supports all Logging levels. Turns ON all logging

NOTE:-

Initially, Apache Software foundation has developed Commons Logging API.

Log4J Framework internally uses Apache Commons Logging which is available in

commons-logging-x.x.jar file.

Official web site for Apache Log4J Framework

http://logging.apache.org/log4j/1.2/download.html

Log4J Frameworks Architecture:

Log4J Framework is broken into three major parts

 Logger

 Formatter

 Handler (Appender)

The Logger is responsible for capturing the message to be logged along with certain metadata and

passing it to the logging framework.

After receiving the message, the framework calls the Formatter with the message.

The Formatter accepts the message object and formats it for output.

The framework then hands the formatted message to the appropriate Appender for disposition.

This might include a console display, writing to disk, appending to a database, or email.

Advantages of Log4j:

– Simpler log level hierarchy and log methods

– More handlers

– Mighty formatters

– Optimized logging costs

– Simpler log level hierarchy:

Steps to integrate Log4J Framework with Java Applications

a) Download log4j-x.x.jar file and add it to your application classpath

b) As per your project requirements, configure logging details in log4j.properties or log4j.xml file

c) User Logger to log messages in your application Java Programs

Configuration File Examples

a) log4j.properties

b) log4j.xml

Simple log4j.properties example

It configures the following things

 ConsoleAppender to display logs to a console

 PatternLayout

 Conversion

 Pattern

log4j.rootLogger= info, way2it

log4j.appender.way2it=org.apache.log4j.ConsoleAppender

log4j.appender.way2it.layout=org.apache.log4j.PatternLayout

log4j.appender. way2it.layout.conversionPattern=%d{yyyy-MM-dd

HH:mm:ss,SSS} %-5p - %m%n

It is using the following conversion pattern

%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p - %m%n

Complex and flexible log4j.properties file configuration

Here it defines the following things

A) To destinations – File and Console

B) Two Appenders

ConsoleAppender

FileAppender

C) Conversion pattern

log4j.rootLogger=INFO,C,F

#log4j.rootLogger=DEBUG,F

log4j.appender.C=org.apache.log4j.ConsoleAppender

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

log4j.appender.F=org.apache.log4j.FileAppender

log4j.appender.F.layout=org.apache.log4j.HTMLLayout

log4j.appender.F.file=banking.html

log4j.appender.F.layout.conversionPattern=%d{yyyy-MM-dd

HH:mm:ss,SSS} %-5p - %m%n

log4j.rootLogger=INFO,C,F

#log4j.rootLogger=DEBUG,F

log4j.appender.C=org.apache.log4j.ConsoleAppender

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

log4j.appender.F=org.apache.log4j.RollingFileAppender

log4j.appender.F.layout=org.apache.log4j.HTMLLayout

log4j.appender.F.file=banking.html

log4j.appender.F.layout.conversionPattern=%d{yyyy-MM-dd

HH:mm:ss,SSS} %-5p - %m%n

#Default is true

log4j.appender.F.append=true

Here it is using RollingFileAppender Appender to log data into a log file.

What is the major difference between FileAppender and RollingFileAppender?

FileAppender:

Whenever we log data to a file, it overwrites the existing log data with new log data.

That means first it cleans the existing log data and add new data

RollingFileAppender:

Whenever we log data to a file, it DOES NOT overwrite the existing log data with new log data.

That means, it simply appends the new logging data to the existing log data.

log4j.rootLogger=INFO,C,F

#log4j.rootLogger=DEBUG,F

log4j.appender.C=org.apache.log4j.ConsoleAppender

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

log4j.appender.F=org.apache.log4j.RollingFileAppender

log4j.appender.F.layout=org.apache.log4j.HTMLLayout

log4j.appender.F.file=banking.html

log4j.appender.F.layout.conversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p - %m%n

log4j.appender.F.MaxFileSize=10KB

# Keep one backup file

log4j.appender.R.MaxBackupIndex=1

#Default is true

log4j.appender.F.append=true

In this example, we are using new properties like file size and file backup

log4j.appender.F.MaxFileSize=5MB

It defines the maximum limit of file size. Here our file size is 5MB. Once it reaches, it moves the

Entire log data into backup file and creates new empty log file.

log4j.appender.R.MaxBackupIndex=1

It specifies how many backup files we want to use to store log data.

log4j.xml example

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE log4j:configuration PUBLIC "-//LOGGER"

"http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd">

<log4j:configuration>

<!--

an appender is an output destination, such as the console or a file;

names of appenders are arbitrarily chosen

-->

<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>

<!--

loggers of category 'org.springframework' will only log messages of level "info" or higher;

if you retrieve Loggers by using the class name (e.g. Logger.getLogger(AClass.class))

and if AClass is part of the org.springframework package, it will belong to this category

-->

<logger name="org.springframework">

<level value="info"/>

</logger>

<!--

everything of spring was set to "info" but for class

PropertyEditorRegistrySupport we want "debug" logging

-->

<logger name="org.springframework.beans.PropertyEditorRegistrySupport">

<level value="debug"/>

</logger>

<logger name="org.acegisecurity">

<level value="info"/>

</logger>

<!-- the root category -->

<root>

<!--

all log messages of level "debug" or higher will be logged, unless defined otherwise

all log messages will be logged to the appender "stdout", unless defined otherwise

-->

<level value="debug" />

<appender-ref ref="stdout" />

</root>

</log4j:configuration>

TTCC

TTCC stands for Time Thread Category Component

TTCC is a message format used by log4j. It uses the following pattern:

%r [%t] %-5p %c %x - %m%n

Mnemonic Description

%r Used to output the number of milliseconds elapsed from the

construction of the layout until the creation of the logging event.

%t Used to output the name of the thread that generated the logging

event.

%p Used to output the priority of the logging event.

%c Used to output the category of the logging event.

%x Used to output the NDC (Nested Diagnostic Context) associated with

the thread that generated the logging event.

%m Used to output the application supplied message associated with the

logging event.

%n Used to output the platform-specific newline character or characters.

%d To display current date and time

Note:-

Log4j has been ported by independent authors to C, C++, Python, Ruby, Eiffel and the much maligned

C#.

Log4J Examples:

Example1)

package com.way2it;

import org.apache.log4j.Logger;

public class HelloWorld

{

static Logger logger = Logger.getLogger("com.way2it.HelloWorld");

static public void main(String[] args)

{

logger.debug("Hello world Log.");

}

}

OR

package com.way2it;

import org.apache.log4j.Logger;

public class HelloWorld

{

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

static public void main(String[] args)

{

logger.debug("Hello world Log.");

}

}

Can we use lower case logging levels in log4j.propertiles file?

Yes we can use

#Log4j properties

log4j.rootLogger=debug,bank

OR

#Log4j properties

log4j.rootLogger=DEBUG,bank

Do we need to use same the following properties file for Log4J or we can

use different file names?

a) log4j.properties

b) log4j.xml

Because Log4J API contains org.apache.log4j .LogManager and defines these two

files as show below.

package org.apache.log4j;

public class LogManager

{

public static final String DEFAULT\_CONFIGURATION\_FILE =

"log4j.properties";

public static final String DEFAULT\_XML\_CONFIGURATION\_FILE = "log4j.xml";

}

If we don’t use one of these files, then we cant get output. We can observe some

warning messages.

import org.apache.log4j.Logger;

public class Sample

{

static Logger log = Logger.getLogger(Sample.class);

public static void main(String[] args)

{

log.info("Hello 1");

log.info("Hello 2");

log.info("Hello 3");

JAVA Means DURGA SOFT

DURGA SOFTWARE SOLUTIONS ,202 HUDA Maitrivanam, Ameerpet , Hyd. Ph: 040-64512786 Page 23

}

}

Output:

log4j:WARN No appenders could be found for logger (Sample).

log4j:WARN Please initialize the log4j system properly.

If we don’t want to use log4j.properties or log4j.xml file, then we can use

BasicConfigurator to

get the log messages with default format.

Default implementation of BasicConfigurator

package org.apache.log4j;

public class BasicConfigurator

{s

public static void configure()

{

Logger root = Logger.getRootLogger();

root.addAppender(new ConsoleAppender(new PatternLayout("%r [%t] %p %c

%x - %m%n")));

}

public static void configure(Appender appender)

{

Logger root = Logger.getRootLogger();

root.addAppender(appender);

}

public static void resetConfiguration()

{

LogManager.resetConfiguration();

}

}

JAVA Means DURGA SOFT

DURGA SOFTWARE SOLUTIONS ,202 HUDA Maitrivanam, Ameerpet , Hyd. Ph: 040-64512786 Page 24

By Default, BasicConfigurator is connected to ConsoleAppender and with basic

pattern

%r [%t] %p %c %x - %m%n

The output contains

 Relative time-the number of milliseconds that elapsed since the start of the

program until the invocation of the logging request2,

 The name of the invoking thread between brackets

 The level of the request

 The logger name

 Finally the message.

Example:

import org.apache.log4j.BasicConfigurator;

import org.apache.log4j.Logger;

public class Sample

{

static Logger log = Logger.getLogger(Sample.class);

public static void main(String[] args)

{

BasicConfigurator.configure();

log.info("Hello 1");

log.info("Hello 2");

log.info("Hello 3");

}

}

Output:

0 [main] INFO Sample - Hello 1

0 [main] INFO Sample - Hello 2

0 [main] INFO Sample - Hello 3

JAVA Means DURGA SOFT

DURGA SOFTWARE SOLUTIONS ,202 HUDA Maitrivanam, Ameerpet , Hyd. Ph: 040-64512786 Page 25

%r [%t] %p %c %x - %m%n

NOTE:-

BasicConfigurator.configure() being the simplest but also the least flexible.

For each Log4J logging levels, Logger class contains respective methods.

Logger class contains the following methods

debug() – DEBUG level

info() – INFO level

warn() – WARN level

error() – ERROR level

fatal() – FATAL level

Ordering of Log4J Log levels:

ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

If we want stop logs in Production or LIVE systems, how do it in Log4J

Framework?

Changing logging level form existing value to OFF in log4j.properties file

log4j.rootLogger=OFF,C,F

log4j.appender.C=org.apache.log4j.ConsoleAppender

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

root DEBUG DEBUG

x ERROR ERROR

x.y none ERROR

x.y.z INFO INFO

Logger name Assigned level Effective

level

root DEBUG DEBUG

x ERROR ERROR

x.y INFO INFO

x.y.z none INFO

BasicConfigurator.configure()

It is equal to the following log4j.properties file

# Set root logger level to DEBUG and add an appender called A1.

log4j.rootLogger=DEBUG, A1

# A1 is set to be a ConsoleAppender.

log4j.appender.A1=org.apache.log4j.ConsoleAppender

If we want to configure properties programmatically, we can do it using

PropertyConfigurator class available in org.apache.log4j file.

log.error("eror 3");

}

}

How to define different logging levels for different Project Modules?

Let us assume that we have the following two modules in our Banking application

a) CreditCard Module

b) DebitCard Module

log4j.properties

# CreditCardFileAppender - used to log messages in the creditcard.log file.

log4j.appender.CreditCardFileAppender=org.apache.log4j.FileAppender

log4j.appender.CreditCardFileAppender.File=creditcard.log

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

log4j.appender.CreditCardFileAppender.layout.ConversionPattern= %-4r [%t] %-5p

%c %x - s%m%n

# DebitCardFileAppender - used to log messages in the debitcard.log file.

log4j.appender.DebitCardFileAppender=org.apache.log4j.FileAppender

log4j.appender.DebitCardFileAppender.File=debitcard.log

[main] FATAL com.way2it.creditcard.CreditCardModule - Sample fatal message

The contents of debitcard.log file.

[main] DEBUG com.way2it.debitcard.DebitCardModule - Sample debug message

[main] INFO com.way2it.debitcard.DebitCardModule Sample info message

[main] WARN com.way2it.debitcard.DebitCardModule - Sample warn message

[main] ERROR com.way2it.debitcard.DebitCardModule - Sample error message

[main] FATAL com.way2it.debitcard.DebitCardModule - Sample fatal message

public class HelloWorld

{

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

public static void main(String[] args)

{

DOMConfigurator.configure("log4j.xml");

logger.debug("Sample debug message");

[main] ERROR com.way2it.helloworld.HelloWorld - Sample error message

[main] FATAL com.way2it.helloworld.HelloWorld - Sample fatal message

Top Eleven tips on logging in Java

1) Use isDebugEnabled() for putting debug log in Java , it will save lot of string

concatenation activity if your code run in production environment with production

logging level instead of DEBUG logging level.

2) Carefully choose which kind of message should go to which level for logging in

Java, It become extremely important if you are writing server application in core

and reset logging in java.

4) By using log4j.xml you can have different logger configuration for different java

classes as well. You can have some classes in INFO mode, some in WARN mode or

ERROR mode. It’s quite flexible to do this to customize java logging.

5) Another important point to remember is format of java logging, this you specify

in logger. Properties file in case of java.util.logging API for logging to use which

java logging Formatter. Don’t forget to include Thread Name and fully qualified java

class Name while printing logs because it would be impossible to find sequence of

events if your code is executed by multiple threads without having thread name on

it. In my opinion this is the most important tips you consider for logging in Java.

6) By carefully choosing format of java logging at logger level and format of writing

log you can have generate reports from your java log files. Be consistent while

logging messages, be informative while logging message, print data with message

whereever required.

7)while writing message for logging in Java try to use some kind of prefix to

indicate which part of your code is printing log e.g. client side , Database site or

session side, later you can use this prefix to do a "grep" or "find" in Unix and have

related logs at once place. Believe me I have used this technique and it helped a lot

while debugging or investigating any issues and your log file is quite large. For

example you can put all Database level log with a prefix "DB\_LOG:" and put all

session level log with prefix "SESSION\_LOG:”

8) If a given logger is not assigned a level, then it inherits one from its closest

ancestor. That’s why we always assign log level to root logger in configuration file

log4j.rootLogger=DEBUG.

9) Both no logging and excessive logging is bad so carefully planned what to log

and on which level you log that messages so that you can run fast in production

environment and at same time able to identify any issue in QA and TEST

environment.

10) I found that for improving logging its important you look through your log and

monitor your log by yourself and tune it wherever necessary. It’s also important to

log in simple English and it should make sense and human readable since support

Uses Stack per user

Most real-world systems have to deal with multiple clients simultaneously. In a

typical multithreaded implementation of such a system, different threads will

handle different clients. Logging is especially well suited to trace and debug

complex distributed applications. A common approach to differentiate the logging

output of one client from another is to instantiate a new separate logger for each

client. This promotes the proliferation of loggers and increases the management

overhead of logging.

To uniquely stamp each request, the user pushes contextual information into the

NDC, the abbreviation of Nested Diagnostic Context. The NDC class is shown below.

public class NDC {

// Used when printing the diagnostic

public static String get();

// Remove the top of the context from the NDC.

public static String pop();

// Add diagnostic context for the current thread.

public static void push(String message);

// Remove the diagnostic context for this thread.

public static void remove();

Yes Log4j is thread-safe.

It's not just thread-safe, it uses synchronized methods everywhere. That means

that for heavy logging log4j will induce performance bottleneck and possible

deadlocks for EJBs.

How and when do loggers inherit level?

Following 3 rules could be applied:

 Root logger always has an assigned level.

 A logger which does not have an explicit log level specified inherits the

parent's log level.

 If the parent logger is not initialized then the parent's parent is checked and

so on, until the root logger.

So , in effect, every logger which does not have an explicit level, will inherit from its

parent. The parent may in turn inherit for its own parent, if no explicit logging was

specified for the parent, and so on. until the root logger, which is guaranteed to

have a log level.

Where can i download log4j?

We can download Log4J Framework at the following log4j website.

http://logging.apache.org/log4j/index.html

A log request is said to be enabled (and logged), if the log level of the request is

higher than or equal to the log level of the logger.

IF LOG.LEVEL >= LOGGER.LEVEL, then the LOG is assumed enabled.

For normal log levels in log4j the ascending log levels are DEBUG, INFO, WARN,

ERROR and FATAL

Also to filter just the messages with a particular level, without the allowing the

higher or lower levels to be logged is done using LevelMatchFilter

How many loggers of the same name can exist?

Only 1 logger with a specified name can exist at a time.

Multiple requests to getLogger API for a same logger name, returns exact same

reference for each of the requests.

SLF4J

What is SLF4J? Why and when do we need to use SLF4J?

SLF4J Stands for Simple Logging Facade for Java

The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or

abstraction for various logging frameworks, e.g. java.util.logging, log4j and

logback, allowing the end user to plug in the desired logging framework at

deployment time.

Similarities and Differences with Log4j

Five of Log4j's six logging levels are used. FATAL has been dropped on the basis

that inside the logging framework is not the place to decide when an application

should terminate and therefore there is no difference between ERROR and FATAL

from the logger's point of view.

Logger instances are created via the LoggerFactory, which is very similar in Log4j.

For example,

private static final Logger LOG = LoggerFactory.getLogger(Wombat.class);

In Logger, the logging methods are overloaded with forms that accept one, two or

more values.[1] Occurrences of the simple pattern {} in the log message is

replaced in turn with the values. This is simple to use yet provides a performance

benefit when the values have expensive toString() methods. When logging is

is disabled, avoiding unnecessary processing.

Unlike Log4j, SLF4J offers logging methods that accept markers. These are special

objects that enrich the log messages and are an idea that SLF4J has borrowed from

logback.