Manual de Log4J

Mauricio Santacruz, VimeWorks Cia. Ltda.

El presente documento es una traducción del manual Short introduction to log4j [http://jakarta.apache.org/log4j/docs/manual.html] publicado originalmente en Inglés bajo los términos de la Apache Software License, además contiene aclaraciones tomadas de los artículos Don't Use System.out.println! Use Log4j [http://www.vipan.com/htdocs/log4jhelp.html], Build Flexible Logs With log4j [http://www.onjava.com/pub/a/onjava/2002/08/07/log4j.html?page=1] y de la experiencia personal del autor.

Table of Contents

Abstract

El presente documento describe el API de log4j, esto únicamente en diseño y modo de utilización. Log4J es un producto open source basado en el trabajo de muchos autores. Este permite a los desarrolladores el controlar la salida de sus mensajes y hacia donde son direccionados con gran granularidad. Es completamente configurable en tiempo de ejecución utilizando archivos externos de configuración. Log4J ha sido implementado en otros lenguajes como: C, C++, C#, Python, Ruby, e Eiffel.

Loggers

La primera y una de las mayores ventajas de cualquier API de logging sobre el tradicional **System.out.println** es la

capacidad de habilitar y deshabilitar ciertos logs, mientras otros no sufren ninguna alteración. Esto se realiza categorizando los mensajes de logs de acuerdo al criterio del programador.

Log4J tiene por defecto 5 niveles de prioridad para los mensajes de logs:

DEBUG: Se utiliza para escribir mensajes de depuración, este log no debe estar activado cuando la aplicación se encuentre en producción.

INFO: Se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.

WARN: Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.

ERROR: Se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.

FATAL: Se utiliza para mensajes críticos del sistema, generalmente luego de guardar el mensaje el programa abortará.

Adicionalmente a estos niveles de log, existen 2 niveles extras que solo se utilizan en el archivo de configuración, estos son:

ALL: este es el nivel más bajo posible, habilita todos los logs.

OFF: este es el nivel más alto posible, deshabilita todos los logs.

Appenders

La posibilidad de selectivamente habilitar y deshabilitar ciertos logs es solo una parte del alcance, log4j permite que los mensajes de logs se impriman en multiples destinos; en Log4J el destino de salida se denomina un appender.

Algunos de los Appenders disponibles son:

ConsoleAppender (org.apache.log4j.ConsoleAppender)

Este appender despliega el log en la consola, tiene las siguientes opciones:

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

Target=System.err: El valor por defecto es **System.out**. Establece la salida del sistema a ser utilizada.

FileAppender (org.apache.log4j.FileAppender)

Este appender redirecciona los mensajes de logs hacia un archivo.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

File=mylog.txt: Nombre del archivo donde se almacenará el log. Se puede utilizar el nombre de algún property (**${nombre_de_la_propiedad}**) para especificar el nombre o la ubicación del archivo.

Append=false: El valor por defecto es **true**, para que los nuevos mensajes de logs se adicionen al archivo existente. Si se especifica **false**, cuando se inicie la aplicación el archivo de log se sobreescribirá.

RollingFileAppender (org.apache.log4j.RollingFileAppender)

Este appender redirecciona los mensajes de logs hacia un archivo, se pueden definir politicas de rotación para que el archivo no crezca indefinidamente.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

File=mylog.txt: Nombre del archivo donde se almacenará el log. Se puede utilizar el nombre de algún property (**${nombre_de_la_propiedad}**) para especificar el nombre o la ubicación del archivo.

Append=false: El valor por defecto es **true**, para que los nuevos mensajes de logs se adicionen al archivo existente. Si se especifica **false**, cuando se inicie la aplicación el archivo de log se sobreescribirá.

MaxFileSize=100KB: Los sufijos pueden ser KB, MB o GB. Rota el archivo de log cuando alcanza el tamaño especificado.

MaxBackupIndex=2: Mantiene un máximo de 2 (por ejemplo) archivos de respaldo. Borra los archivos anteriores. Si se especifica **0** no se mantiene respaldos.

DailyRollingFileAppender (org.apache.log4j.DailyRollingFileAppender)

Este appender redirecciona los mensajes de logs hacia un archivo, se pueden definir politicas de rotación basados en la fecha.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

File=mylog.txt: Nombre del archivo donde se almacenará el log. Se puede utilizar el nombre de algún property (**${nombre_del_property}**) para especificar el nombre o la ubicación del archivo.

Append=false: El valor por defecto es **true**, para que los nuevos mensajes de logs se adicionen al archivo existente. Si se especifica **false**, cuando se inicie la aplicación el archivo de log se sobreescribirá.

DatePattern='.'yyyy-ww: Rota el archivo cada semana Se puede especificar que la frecuencia de rotación sea mensual, semanal, diaria, 2 veces al día, cada hora o cada minuto. Este valor no solo especifica la frecuencia de rotación sino el sufijo del archivo de respaldo. Algunos ejemplos de frecuencia de rotación son:

'.'yyyy-MM: Rota el archivo el primero de cada mes
'.'yyyy-ww: Rota el archivo al inicio de cada semana
'.'yyyy-MM-dd: Rota el archivo a la media noche todos los días
'.'yyyy-MM-dd-a: Rota el archivo a la media noche y al media día, todos los días
'.'yyyy-MM-dd-HH: Rota el archivo al inicio de cada hora
'.'yyyy-MM-dd-HH-mm: Rota el archivo al inicio de cada minuto

SocketAppender (org.apache.log4j.net.SocketAppender)

Redirecciona los mensajes de logs hacia un servidor remoto de log.

El SocketAppender tiene las siguientes propiedades:

El mensaje de log se almacenará en el servidor remoto sin sufrir alteraciones en los datos (como fecha, tiempo desde que se inicio la aplicación, NDC), exactamente como si hubiese sido guardado localmente.
El SocketAppender no utiliza Layout. Únicamente serializa el objeto LoggingEvent para enviarlo.
Utiliza TCP/IP, consecuentemente si el servidor es accesible el mensaje eventualemente arribará al servidor.
Si el servidor remoto está abajo, la petición de log será simplemente rota. Cuando el servidor vuelva a estar disponible la transmisión se reasume transparentemente.Esta reconexión transparente es realizada por un connector thread que periodicamente revisa si existe conexión con el servidor.
Los eventos de logging son automaticamente almacenados en memoria por la implementación nativa de TCP/IP. Esto quiere decir que si la conexión hacia el servidor de logs es lenta y la conexión con los clientes es rápida, los clientes no se ven afectados por la lentitud de la red.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

port=6548: Especifica el puerto por el que se va a comunicar al servidor de logs.

remoteHost=192.168.10.225: Especifica la máquina donde se encuentra el servidor de logs.

reconnectionDelay=300: El valor por defecto es de 30000 milisegundos, que corresponden a 30 segundos. Es un valor entero que corresponde al número de milisegundos que va a esperar para volver a intentar conectarse con el servidor cuando se ha perdido comunicación con este.

Log4J provee 2 implementaciones de un servidor de logs, estas implementaciones son:

1. **SocketServer**

   Utilización: **java org.apache.log4j.net.SocketServer port configFile configDir**

   donde **port** es el número de puerto por el que va a escuchar el servidor, **configFile** es el archivo de configuración (debe ser un Java Properties) y **configDir** es la ruta hacia el directorio que contiene los archivos de configuración, posiblemente uno por cada cliente.

   configFile es utilizado para configurar la jerarquia por defecto del SocketServer.

   Cuando una conexión es abierta por un nuevo host, digamos foo.bar.net, entonces el SocketServer busca un archivo de configuración llamado foo.bar.net.lcf bajo el **configDir** que es pasado como el tercer argumento. Si el archivo es encontrado entonces una nueva jerarquía es configurada utilizando este archivo. Si el mismo host realiza otra petición, se utiliza la jerarquía previamente configurada.

   En caso que el archivo foo.bar.net-lcf no existe se utiliza el archivo de configuración genérico que se llama generic.lcf bajo el directorio **configDir**. Si este arhcivo no existe se utiliza la jerarquía por defecto.

   El tener diferentes clientes utilizando diferentes jerarquías asegura una total independencia de los clientes con respecto al servidor de log.

   Actualmente el SocketServer utiliza las jerarquías de acuerdo a la dirección IP del equipo o a su nombre, por lo que si dos aplicaciones separadas están corriendo en el mismo equipo, las dos compartirán la misma jerarquía.

2. **SimpleSocketServer**

   Utilización: **java org.apache.log4j.net.SimpleSocketServer port configFile**

   donde **port** es el número de puerto por el que va a escuchar el servidor y **configFile** es el archivo de configuración (puede ser Java Properties o XML).

SMTPAppender (org.apache.log4j.net.SMTPAppender)

Envia un mail con los mensajes de logs, típicamente se utiliza para los niveles ERROR y FATAL.

El número de eventos de log enviados en el mail depende del valor de BufferSize. El SMTPAppender mantiene únicamente el último BifferSize en un buffer cíclico.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

To=direccion1@dominio.com: Especifica la dirección de correo a donde se enviará el mail.

From=direccion2@dominio.com: Especifica la dirección de correo desde donde se envia el mail.

SMTPHost=mail.dominio.com: Especifica el servidor de SMTP que se va a utilizar para enviar el mail.

Subject=Mensajes de Logs: Especifica el asunto del mail.

LocationInfo=true: El valor por defecto es **false**. Envía información acerca de la ubicación donde se generó el evento de log.

BufferSize=20: El valor por defecto es 512. Representa el número máximo de eventos de logs que serán recolectados antes de enviar el mail.

JDBCAppender (org.apache.log4j.jdbc.JDBCAppender)

ALERTA: Esta versión de JDBCAppender será reemplazada en el fututo.

Este appender redirecciona los mensajes de log hacia una base de datos.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

Driver=mm.mysql.Driver: Define el driver a ser utilizado, asegurese de tener la clase en el CLASSPATH.

URL=jdbc:mysql://localhost/LOG4JDemo: Es la cadena de conexión que se utilizará.

user=default: El usuario de la base de datos.

password=default: La clave para poder ingresar a la base de datos.

sql=INSERT INTO JDBCTEST (Message) VALUES ('%d - %c - %p - %m'): La instrucción SQL que se utiliza para grabar el mensaje en la base de datos. Si se desea se puede guardar cada valor en columnas separadas,

por ejemplo: INSERT INTO JDBCTEST (Date, Logger, Priority, Message) VALUES ('%d', '%c', '%p', '%m').

SyslogAppender (org.apache.log4j.net.SyslogAppender)

Este appender redirecciona los mensajes de logs hacia el demonio remoto syslog en sistemas operativos Unix.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

Facility=USER: Los valores permitidos en este parámetro son: KERN, USER, MAIL, DAEMON, AUTH, SYSLOG, LPR, NEWS, UUCP, CRON, AUTHPRIV, FTP, LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7.

FacilityPrinting=true: El valor por defecto es **false**. Si se encuentra habilitado en el mensaje enviado se adicionará el Facility Name.

SyslogHost=localhost: Es el nombre del equipo hacia donde serán enviados los mensajes. **ALERTA:** Si este parámetro no es colocado, el appender falla.

NTEventLogAppender (org.apache.log4j.nt.NTEventLogAppender)

Este appender redirecciona los mensajes de logs hacia los logs del sistema de NT.

ALERTA: Este appender solo puede ser intalado y utilizado en sistemas Windows.

No olvide colocar el archivo NTEventLogAppender.dll en un directorio que se encuentre en el PATH de Windows. De otra manera obtendrá un error java.lang.UnsatisfiedLinkError.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es **true**, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

Source=Nombre_de_la_Aplicación: El valor por defecto es Source. Especifica el nombre con el que será especificado el evento.

JMSAppender (org.apache.log4j.net.JMSAppender)

Este appender serializa los eventos y los transmite como un mensaje JMS tipo ObjectMessage.

Threshold=WARN: Este parámetro establece que el appender no despliega ningún mensaje con prioridad menor a la especificada aquí.

ImmediateFlush=true: El valor por defecto es true, esto quiere decir que los mensajes de log no son almacenados en un buffer, sino que son enviados directamente al destino.

LocationInfo=true: El valor por defecto es false. Envía información acerca de la ubicación donde se generó el evento de log.

InitialContextFactoryName=org.jnp.interfaces.NamingContextFactory: Especifica el InitialContextFactoryName para la conexión por JNDI.

ProviderURL=jnp://hostname:1099: Especifica el URL del servidor al cual se enviarán los mensajes.

URLPkgPrefixes=org.jboss.naming:org.jnp.interfaces: Especifica el URLPkfPrefixes para la conexión por JNDI

UserName=usuario: Especifica el nombre usuario si el servidor tiene autentificación.

Password=clave: Especifica la clave si el servidor tiene autentificación.

Layouts

El layout es responsable de formatear los mensajes de logs de acuerdo a las deficiones del desarrollador.

Los tipos de Loyouts disponibles son:

SimpleLayout

Consiste de la prioridad del mensaje, seguido por " - " y luego el mensaje de log. Por ejemplo:

DEBUG - Hello world

PatternLayout

Especifica el formato de salida de los mensajes de logs de acuerdo a unos patrones de conversión similares a los utilizados en la función printf en lenguaje C. Los patrones disponibles son:

Table 1.

Caracter de Conversión	Efecto
c	Utilizado para desplegar la categoría del evento de log. Este caracter de conversión puede tener opcionalmente precisión, que es una constante encerrada en llaves. Si la precisión no es especificada, se imprimirá el nombre completo de la categoría. Por ejemplo para el nombre de categoría "a.b.c" y el patrón **%c{2}** la salida sería "b.c".
C	Utilizado para desplegar el nombre completo de la clase que generó el evento de log. Este caracter de conversión puede tener opcionalmente precisión, que es una constante encerrada en llaves. Si la precisión no es especificada, se imprimirá el nombre completo de la clase. Por ejemplo: para el nombre de clase "com.vimeworks.xyz.AlgunaClase" y el patrón **%C{1}** la salida sería "AlgunaClase". **ALERTA:** Generar llamadas a la información de la clase es lento. Esto no debe ser utilizado a menos que la velocidad de ejecución no sea tomada en cuenta.

Caracter de Conversión	Efecto
d	Utilizado para desplegar la fecha del evento de log. El formato en que se despliege la fecha puede ser especificado entre llaves. Por ejemplo: **%d{HH:mm:ss,SSS}** o **%d{dd MMM yyyy HH:mm:ss,SSS}**. Si no hay un formato especificado, se utiliza el ISO8601. El especificador del formato admite la misma sintaxis que `SimpleDateFormat`. Para mejores resultados es recomendable utilizar los formatos proporcionados por log4j. Estos se pueden utilizar especificando su nombre: **%d{ABSOLUTE}** que da como resultado "HH:mm:ss,SSS"; **%d{DATE}** que da como resultado "dd MMM YYYY HH:mm:ss,SSS" e **%d{ISO8601}** que da como resultado "YYYY-mm-dd HH:mm:ss,SSS".
F	Utilizado para desplegar el nombre del archivo que generó el evento de log. **ALERTA:** Generar llamadas a la información de archivos es extremadamente lento. Esto no debe ser utilizado a menos que la velocidad de ejecución no sea tomada en cuenta.
l	Utilizado para desplegar la información acerca de la ubicación que ha generado el evento de logging. La informacón acerca de la ubicación depende de la implementación del JVM pero usualmente consiste de el nombre completo de la clase, el nombre del método y entre comillas el nombre del archivo y el número de línea. Se le puede utilizar como abreviación para **%C.%M(%F:%L)**. **ALERTA:** La información de la ubicación puede ser útil pero el generarla es extremadamente lento. Esto no debe ser utilizado a menos que la velocidad de ejecución no sea tomada en cuenta.
L	Utilizado para desplegar el número de línea donde se ha generado el evento de logging. **ALERTA:** Generar llamadas a la información de la ubicación es extremadamente lento. Esto no debe ser utilizado a menos que la velocidad de ejecución no sea tomada en cuenta.
m	Utilizado para desplegar el mensaje asociado con el evento de logging.
M	Utilizado para desplegar el nombre del método que generó el evento de logging.

Caracter de Conversión	Efecto
	ALERTA: Generar llamadas a la información de la ubicación es extremadamente lento. Esto no debe ser utilizado a menos que la velocidad de ejecución no sea tomada en cuenta.
n	Despliega el caracter o caracteres de salto de línea dependiendo de la plataforma. Este caracter de conversión ofrece practicamente el mismo rendimiento que utilizar un caracter no portable de salto de línea como es "\n" o "\r\n". Por lo que esta es la vía preferida para especificar el caracter de salto de línea.
p	Utilizado para desplegar la prioridad del evento que gerenó logging.
r	Utilizado para desplegar el número de milisegundos transcurridos desde que comenzó la aplicaión hasta que el eventro de log fue disparado.
t	Utilizado para desplegar el nombre del thread que generó el evento de logging.
x	Utilizado para desplegar el NDC (Nested Diagnostic Context) asociado con el thread que generó el evento de logging.
X	Utilizado para desplegar el MDC (Mapped Diagnostic Context) asociado con el thread que generó el evento de logging. Este caracter de conversión debe ser siempre seguido del "key" del mapa entre corchetes. Por ejemplo **%X{numeroCliente}** donde numeroCliente es el "key". El valor en el MCD correspondiente al "key" será desplegado.
%	La secuencia %% despliega un signo de porcentaje.

Adicionalmente es posible cambiar el ancho máximo y mínimo de cada campo, así como también la justificación.

El modificador opcional se coloca entre el signo de porcentaje y el caracter de conversión.

El primer modificador de formato es la justificación a la izquierda que es representado por el signo menos (-). Luego va el ancho mínimo del campo. Esta es una constante decimal que representa el mínimo número de caracteres que serán desplegados. Si el dato tiene menos caracteres, este es completado a la derecha o a la izquierda hasta completar el ancho mínimo. Por defecto se rellenará a la izquierda (alineado a la derecha) pero también se puede rellenar a la derecha con justificación a la izquierda. El caracter de relleno es el espacio en blanco. Si el dato es más grande que el ancho mínimo, el campo es extendido para acomodar el dato. El valor nunca es truncado.

Esta conducta puede ser cambiada utilizando el ancho máximo del campo que es representada por un punto seguido de una constante decimal. Si el dato es más largo que el valor especificado aquí, los caracteres extras serán

removidos del principio del dato y no del final. Por ejemplo si el ancho máximo está fijado en 8 y el dato contiene 10 caracteres, entonces los 2 primeros caracteres son borrados.

Los siguientes son algunos ejemplos de modificadores de formato:

Table 2.

Modificador de formato	Justificación a la Izquierda	Ancho mínimo	Ancho máximo	Comentario
%20c	falso	20	ninguno	Rellena a la izquierda con espacios en blanco si el nombre de la categoría tiene menos de 20 caracteres.
%-20c	verdadero	20	ninguno	Rellena a la derecha con espacios en blanco si el nombre de la categoría tiene menos de 20 caracteres.
%.30c	ND	ninguno	30	Trunca desde el inicio si el nombre de la categoría es mayor que 30 caracteres
%20.30c	falso	20	30	Rellena a la izquierda con espacios en blanco si el nombre de la categoría tiene menos de 20 caracteres. Y trunca desde el inicio si el nombre de la categoría es mayor que 30 caracteres
%-20.30c	verdadero	20	30	Rellena a la derecha con espacios en blanco si el nombre de la categoría tiene menos de 20 caracteres. Y trunca desde el inicio si el nombre de la categoría es mayor que 30 caracteres

Los siguientes son algunos ejemplos de modificadores de formato:

El siguiente es un ejemplo de patrones de conversión:

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

Este es escencialmente el TTCC layout.

HTMLLayout

Especifica que la salida de los eventos será en una tabla HTML.

LocationInfo=true: El valor por defecto es **false**. Despliega el nombre del archivo JAVA y el número de línea.

Title=Título de mi aplicación: El valor por defecto es "Log4J Log Messages". Especifica el valor que se incluirá en el tag <title> de HTML

XMLLayout

Especifica que la salida será a un archivo XML que cumple con el log4j.dtd.

LocationInfo=true: El valor por defecto es false. Despliega el nombre del archivo JAVA y el número de línea.

TTCCLayout

Consiste de la fecha, thread, categoría y NDC, cualquiera de estos cuatro campos pueden ser habilitados y deshabilitados individualmente.

DateFormat=ISO8601: Define el formato de la fecha a ser utilziado. Se pueden utilzar los siguientes valores: NULL, RELATIVE, ABSOLUTE, DATE o ISO8601.

TimeZoneID=GMT-8:00: Define el formato de la hora a ser utilizado. El parámetro es una cadena de caracteres en el fomato esperado por el método `TimeZone.getTimeZone(java.lang.String)`

CategoryPrefixing=false: El valor por defecto es **true**. Despliega el nombre de la categoría.

ContextPrinting=false: El valor por defecto es **true**. Despliega la información del NDC relacionado con el thread.

ThreadPrinting=false: El valor por defecto es **true**. Despliega el nombre del thread.

ALERTA: No utilice la misma instancia de TTCCLayout con diferentes appenders. El TTCCLayout no está en modo seguro (thread safe).

Configuración

Log4J provee un ambiente totalmente configurable por programación. Aunque la manera más flexible para esto es por medio de archivos de configuración, actualmente estos archivos pueden ser escritos en formato XML o en Java Properties (key=value).

En el siguiente ejemplo la clase MyApp primero importa las clases relacionadas de log4j, luego define una variable estática de tipo Logger con el nombre de la clase MyApp.class y finalmente invoca al método BasicConfigurator.configure que crea una configuración de log4j con un ConsoleAppender y define el PatternLayout a "%-4r [%t] %-5p %c %x - %m%n".

```
import com.foo.Bar;

// Importa las clases de log4j.

import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

public class MyApp {

  // Define una variable estática que tiene una referencia
  // a una instancia de Logger llamada "MyApp".
  static Logger logger = Logger.getLogger(MyApp.class);
```

```
  public static void main(String[] args) {

    // Crea una simple configuración a la consola
    BasicConfigurator.configure();
    logger.info("Entrando en la aplicación.");
    Bar bar = new Bar();
    bar.doIt();
    logger.info("Saliendo de la aplicación.");
  }
}
```

MyApp utiliza la clase Bar del paquete com.foo, la cual ya no tiene que configurar el log sino únicamente utilizarlo.

```
package com.foo;
import org.apache.log4j.Logger;

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

  public void doIt() {
    logger.debug("Dentro de Bar!");
  }
}
```

La salida en el log sería:

0 [main] INFO MyApp - Entrando en la aplicación.

36 [main] DEBUG com.foo.Bar - Dentro de Bar!

51 [main] INFO MyApp - Saliendo de la aplicación.

En el ejemplo anterior no se puede parametrizar el log en tiempo de ejecución, afortunadamente esto es facil de solucionar:

```
import com.foo.Bar;

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

public class MyApp {

  static Logger logger = Logger.getLogger(MyApp.class.getName());

  public static void main(String[] args) {


    // BasicConfigurator es reemplazodo por PropertyConfigurator.
    PropertyConfigurator.configure(args[0]);

    logger.info("Entrando en la aplicación.");
    Bar bar = new Bar();
    bar.doIt();
    logger.info("Saliendo de la aplicación.");
  }
}
```

Como se muestra el **BasicConfigurator** es cambiado por **PropertyConfigurator** que consume el nombre del archivo tipo Properties, también se pudo haber utilizado el **DOMConfigurator** del paquete **org.apache.log4j.xml** que consume archivos en formato XML.

Estos son algunos ejemplos de archivos de configuración:

```
# Coloca el nivel del root logger en DEBUG y adiciona un solo appender que es A1.
log4j.rootLogger=DEBUG, A1

# A1 es configurado para utilizar ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 utiliza PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

Aquí la configuración es igual a **BasicConfigurator**.

```
log4j.rootLogger=DEBUG, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout

# Imprime la fecha en formato ISO 8601
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

# Imprime solo mensajes de nivel WARN o superior en el paquete com.foo.
log4j.logger.com.foo=WARN
```

Esta sería la salida de log con el archivo de configuración anterior:

2000-09-07 14:07:41,508 [main] INFO MyApp - Entrando en la aplicación.

2000-09-07 14:07:41,529 [main] INFO MyApp - Saliendo de la aplicación.

El nivel del paquete com.foo es WARN y el paquete com.foo.Bar tiene un mensaje de log en nivel DEBUG que es menor que el nivel configurado para ese paquete por lo tanto ese mensaje no se despliega.

```
log4j.rootLogger=debug, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# En el log de la consola se desplegará el nombre del archivo y el número de línea.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log

log4j.appender.R.MaxFileSize=100KB
# Mantiene un archivo de respaldo
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

La salida en consola con el archivo de configuración anterior sería:

INFO [main] (MyApp2.java:12) - Entrando en la aplicación.

DEBUG [main] (Bar.java:8) -Dentro de Bar!

INFO [main] (MyApp2.java:15) - Saliendo de la aplicación.

Adicionalmente a este log también se almacena en un archivo llamado `example.log` que es rotado cada vez que

alcanza los 100KB y se almacena una copia de respaldo moviendolo automaticamente al nombre `example.log.1`.

Adicionalmente se pueden utilizar **PropertyConfigurator.configureAndWatch** o **DOMConfigurator.configureAndWatch**, con estos métodos se configura Log4J y se define que se revisará el archivo de configuración por si existe algún cambio.

Inicialización en Servlet

Este es un ejemplo de un Servlet de inicialización donde se configura Log4J:

```
import java.io.File;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Importa las clase de log4j
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class SetupServlet extends HttpServlet {

    /** Initializa el servlet.
     */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);

        // Lee el directorio donde va a ser colocado el archivo de logs
        String directory = getInitParameter("log-directory");

        // Adiciona el parametro del directorio como un Property del sistema
        // para que pueda ser utilizado dentro del archivo de configuración del Log4J
        System.setProperty("log.directory",directory);

        // Extrae el path donde se encuentra el contexto
        // Asume que el archivo de configuración se encuentra en este directorio
        String prefix =  getServletContext().getRealPath("/");

        // Lee el nombre del archivo de configuración de Log4J
        String file = getInitParameter("log4j-init-file");

        if(file == null || file.length() == 0 ||
                !(new File(prefix+file)).isFile()){
                System.err.println(
                "ERROR: No puede leer el archivo de configuración. ");
                throw new ServletException();
        }


        // Revisa otra parámetro de configuración que le indica
        // si debe revisar el archivo de log por cambios.
        String watch = config.getInitParameter("watch");

        // Extrae el parámetro que le indica cada que tiempo debe revisar el archivo de configuració
        String timeWatch = config.getInitParameter("time-watch");

        // Revisa como debe realizar la configuración de Log4J y llama al método adecuado
        if (watch != null && watch.equalsIgnoreCase("true")) {
            if (timeWatch != null) {
                PropertyConfigurator.configureAndWatch(prefix+file,Long.parseLong(timeWatch));
            } else {
                PropertyConfigurator.configureAndWatch(prefix+file);
            }
        } else {
            PropertyConfigurator.configure(prefix+file);
        }
```

```
    }

    /** Destruye el servlet.
     */
    public void destroy() {
        super.destroy();
    }
}
```

La especificación en el archivo web.xml sería la siguiente:

```
  <servlet>
    <servlet-name>log4j-init</servlet-name>
    <servlet-class>com.vimeworks.jpresto.webapp.SetupServlet</servlet-class>
    <init-param>
      <param-name>log-directory</param-name>
      <param-value>/usr/local/tomcat/logs</param-value>
    </init-param>
    <init-param>
      <param-name>log4j-init-file</param-name>
      <param-value>log4j.properties</param-value>
    </init-param>
    <init-param>
      <param-name>watch</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>time-watch</param-name>
      <param-value>10000</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

Nested Diagnostic Context / Mapped Diagnostic Context

El Nested Diagnostic Context o NDC, es un instrumento para distinguir los mensajes de logs que son intercalados en un solo lugar de diferentes fuentes. Los mensajes de logs son típicamente intercalados cuando un servidor maneja varios clientes de forma simultanea.

La forma de reconocer logs intercalados es por medio de una marca en el mensaje, es allí donde NDC juega un papel importante.

NDC es manejado uno por cada thread, esto quiere decir que todas operaciones como **push**, **pop()**, **clear()**, **getDepth()** y **setMaxDepth(int)** afectan únicamente al thread actual, el NDC de otros threads no sufre ninguna alteración.

Por ejemplo un Servlet puede crear un NDC por cada request, este puede consistir del host name del cliente y cualquier otra información particular de éste. Para crear el NDC utiliza la función **push**. Tome en cuenta los siguientes puntos: La gran obligación es llamar al método remove() cuando se salga del thread. Ya que ahí

Los contextos pueden ser anidados.
Cuando entre a un contexto llame a **NDC.push**. Si no existe un NDC para este thread, este método lo crea.
Cuando salga de un contexto utilice **NDC:pop**.
Cuando salga del thread esté seguro de llamar a NDC.remove().
siguientes puntos: La gran obligación es llamar al método remove() cuando se salga del thread. Ya que ahí aseguramos que esta memoria será liberada por el Java garbage collector.

Un thread puede heredar el NDC de otro thread (posiblemente el padre) usando el método **inherit**. Un thread puede obtener una copia del NDC con el método **cloneStack** y pasar esta referencia a otro thread, particularmente un hijo.

Estos son los métodos disponibles para NDC:

Table 3.

Valor de Retorno	Método	Información
static void	clear()	Elimina cualquier información de NDC
static Stack	cloneStack()	Clona el NDC del thread actual
static String	get()	Nunca utilice este método directamente. Utilice el método LoggingEvent.getNDC()
static int	getDepth()	Retorna el nivel de profundidad de anidamiento para el NDC actual
static void	inherit(Stack stack)	Hereda el NDC de otro thread
static String	peek()	Retorna el valor del tope de la pila sin removerlo
static String	pop()	Los clientes pueden llamar a este método antes de cambiar a otro contexto.
static void	push(String message)	Coloca un nuevo mensaje en el NDC del thread actual
static void	remove()	Remueve el NDC del thread actual
static void	setMaxDepth(int maxDepth)	Determina el número máximo de anidamiento

Estos son los métodos disponibles para NDC:

La clase MDC es similar al NDC excepto que está basado en un mapa y no en una pila. Esto provee Mapped Diagnostic Contexts.

La clase MDC requiere JDK 1.2 o superior. Bajo JDK 1.1 el MDC siempre retorna valores vacios, pero la aplicación no es afectada.

Estos son los métodos diponibles para MDC:

Table 4.

Valor de Retorno	Método	Información
static Object	get(String key)	Retorna el contexto identificado por el parámetro key
static HashTable	getContext()	Retorna el MDC del thread actual como un hastable
static void	push(String key, Object o)	Coloca un valor de contexto (el parámetro o) identificado por el parámetro key dentro del MDC del thread actual
static void	remove(String key)	Remueve el valor de contexto

Valor de Retorno	Método	Información
		identificado por el parámetro key

Rendimiento

Uno de los argumentos utilizados en contra de realizar logging es el costo computacional. Esta es una preocupación legitima ya que en aplicaciones de tamaño moderado el pedido de logs puede causar sobrecarga. El objetivo principal de Log4J es la velocidad y luego la flexibilidad.

Los usuarios deben tener en mente los siguientes puntos:

1. **El rendimiento del sistema cuando el logging está desactivado.**

   Cuando el log se encuentra desactivado el costo de la llamada al log consiste de la invocación de un método más la comparación de un entero. En un equipo Pentium II de 233 MHz el costo se encuentra en el rango de 5 a 50 nanosegundos.

   Generalmente la invocación del método contiene el costo "escondido" al construir los parámetros. Por ejemplo:

   ```
   logger.debug("Entry number: " + i + " is " + String.valueOf(entry[i]));
   ```

   En este caso se incurre en el costo de construir los parámetros del mensaje. Convertir ambos integer i y entry[i] a String, y concatenar cadenas intermedias, todo esto para que el mensaje no sea desplegado. El costo de construir los parámetros puede ser alto dependiendo del tamaño y los parámetros involucrados.

   Para reducir el costo de la construcción de parámetros utilice:

   ```
   if(logger.isDebugEnabled() {
       logger.debug("Entry number: " + i + " is " + String.valueOf(entry[i]));
   }
   ```

   En este código no se incurre en el costo de procesar los parámetros innecesariamente. De otro lado si el log se encuentra habilitado la sobrecarga es insignificante ya que es un aumento de alrededor del 1% de lo que toma realizar el log normalmente.

2. **El rendimento cuando tiene que decidir si el nivel de log se encuentra activado o no**

   Escencialmente el rendimiento no es afectado ya que los loggers son jerarquicos. El costo típico de comparar en forma jerárquica es 3 veces menor que cuando el log es desactivado completamente.

3. **Formateo de los mensajes**

   Este es el costo de formatear la salida y enviarlo al destino adecuado. Este no es un problema ya que los loggers como los appenders son optimizados. El costo de enviar un log es alrededor de 100 a 300 microsegundos.