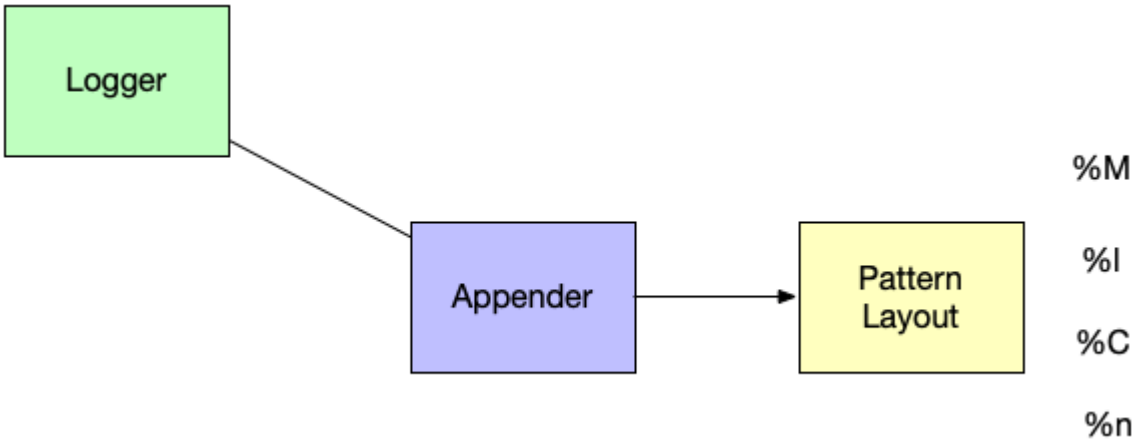
La configuración de Log4J PatternLayout es una de las configuraciones más habituales que a un desarrollador le puede tocar modificar cuando tenemos un problema determinado . Un PatternLayout como su nombre indica sirve para definir el formato de patronaje con el que el API de Log4J emite los mensajes ya sea a consola , fichero, base de datos o otro sistema.



Vamos a ver varios ejemplos que nos ayuden a entender las cosas.

```
package com.arquitecturajava.log4j;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class PrincipalLog {

        public static void main(String[] args) {
                Logger milogger= LogManager.getLogger("milogger");
                milogger.trace("hola desde mi logger traza");
                milogger.debug("hola desde mi logger debug");
                milogger.info("hola desde mi logger info");
                milogger.warn("hola desde mi logger warn");
                milogger.error("hola desde mi logger error");
                milogger.fatal("hola desde mi logger fatal");
```

```
                System.out.println("*************************");
        }
}
```

```
 hola desde mi logger traza
 hola desde mi logger debug
 hola desde mi logger info
 hola desde mi logger warn
 hola desde mi logger error
 hola desde mi logger fatal
************************
```

Estamos ante el ejemplo más sencillo de log en el cual somos nosotros los que emitimos los mensajes con la información del nivel o categoría en el que están. Si revisamos el fichero de log de log4j2 tenemos una configuración de emisión de mensajes sobre la consola:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="logConsola" target="SYSTEM_OUT">
        </Console>
    </Appenders>
    <Loggers>
        <Root level="trace">
           <AppenderRef ref="logConsola"/>
        </Root>
    </Loggers>
</Configuration>
```

Todo es muy sencillo , vamos a realizar el primer cambio y aplicar un pattern layout con dos parámetros:

- %m: Imprime el mensaje de traza

- %n: imprime un retorno de linea

```
   <Console name="logConsola" target="SYSTEM_OUT">
         <PatternLayout
                pattern=" %m%n" />
        </Console>
```

Acabamos de modificar la forma en la que Log4J PatternLayout se imprime . Si nosotros ademas modificamos los mensajes para que sean los más sencillos posibles con:

```
package com.arquitecturajava.log4j;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class PrincipalLog2 {

        public static void main(String[] args) {
                Logger milogger= LogManager.getLogger("milogger");
                milogger.trace("hola desde mi logger ");
                milogger.debug("hola desde mi logger ");
                milogger.info("hola desde mi logger ");
                milogger.warn("hola desde mi logger ");
                milogger.error("hola desde mi logger ");
                milogger.fatal("hola desde mi logger ");
                System.out.println("************************");
        }
}
```

El resultado que vemos salir en la consola es :

```
 hola desde mi logger
 hola desde mi logger
```

```
 hola desde mi logger
 hola desde mi logger
 hola desde mi logger
 hola desde mi logger
************************
```

Trazas y niveles

Como podemos observar perdemos la información de que nivel de trata tenemos . Eso tiene su lógica ya que hemos cambiado los mensajes y eliminado esa información que escribíamos nosotros a mano. Es momento de modificar Log4j patternLayout y añadir el nivel de traza con %p que significa prioridad.

```
    <Console name="logConsola" target="SYSTEM_OUT">
         <PatternLayout
                pattern="%p %m%n" />
        </Console>
```

Con esta información si volvemos a ejecutar el programa la cosa cambia bastante:

```
TRACE hola desde mi logger
DEBUG hola desde mi logger
INFO hola desde mi logger
WARN hola desde mi logger
ERROR hola desde mi logger
FATAL hola desde mi logger
************************
```

Ahora sí nos aparece el nivel de traza en cada mensaje o como Log4j lo denomina, el nivel de prioridad del mensaje.

Log4J PatternLayout y Categorías

De igual forma que deseamos saber el nivel del mensaje , es imprescindible saber que clase o package genero el mensaje. Para ello usaremos %C que nos dirá la categoría exacta del mensaje.

```
<Console name="logConsola" target="SYSTEM_OUT">
         <PatternLayout
                pattern="%C %p  %m%n" />
        </Console>
```

Realizado este cambio la mensajería de log cambia:

```
com.arquitecturajava.log4j.PrincipalLog2 TRACE  hola desde mi logger
com.arquitecturajava.log4j.PrincipalLog2 DEBUG  hola desde mi logger
com.arquitecturajava.log4j.PrincipalLog2 INFO  hola desde mi logger
com.arquitecturajava.log4j.PrincipalLog2 WARN  hola desde mi logger
com.arquitecturajava.log4j.PrincipalLog2 ERROR  hola desde mi logger
com.arquitecturajava.log4j.PrincipalLog2 FATAL  hola desde mi logger
************************
```

Podemos ver la clase especifica que genera el log de una forma clara.

PatternLayouts y Fechas

Otro dato muy importante a la hora de manejar una mensajería de log es ser muy consciente de cuando se ha generado ese mensaje . Para ello podemos usar %d que define la fecha aplicandole un pequeño subformato que clarifique.

```
<Console name="logConsola" target="SYSTEM_OUT">
         <PatternLayout
                pattern=" %d{yyyy-MM-dd HH:MM:ss} %C %p  %m%n" />
```

```
        </Console>
```

En este caso el patrón define el año el mes la fecha y la hora en Log4J

El resultado es :

```
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 TRACE
hola desde mi logger
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 DEBUG
hola desde mi logger
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 INFO
hola desde mi logger
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 WARN
hola desde mi logger
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 ERROR
hola desde mi logger
 2022-07-27 10:07:16 com.arquitecturajava.log4j.PrincipalLog2 FATAL
hola desde mi logger
************************
```

De esta manera la información queda todavía mucho más clara.

Afinando Log4J PatternLayout

Puede ser que queramos ver las cosas todavía más a detalle existen varias opciones que ayudan a clarificar más las cosas:

- %l : La linea en la cual se ha generado el log
- %M : El metodo que ha generado el log

```
 <Console name="logConsola" target="SYSTEM_OUT">
        <PatternLayout
               pattern="  %d{yyyy-MM-dd HH:MM:ss}   %l %p  %m%n" />
```

```
        </Console>
```

Eliminamos %C para que no salga todo repetido y añadimos %l con la linea por ejemplo:

```
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:12)
TRACE  hola desde mi logger
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:13)
DEBUG  hola desde mi logger
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:14)
INFO  hola desde mi logger
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:15)
WARN  hola desde mi logger
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:16)
ERROR  hola desde mi logger
  2022-07-27 10:07:15  main
com.arquitecturajava.log4j.PrincipalLog2.main(PrincipalLog2.java:17)
FATAL  hola desde mi logger
************************
```

Muestra la información de una forma mucho más clara y útil para el desarrollador.

Conclusiones

Todos necesitamos tener un conocimiento básico de Log4J PatternLayouts y los patrones más básicos que normalmente se implementan.

Otros artículos relacionados

- Spring Boot JPA y su configuración
- ¿Qué es Gradle?
- Spring Boot Load Data y testing
- Log4j