LOG4J

MARTINEZ CRUZ, ALVARO JAVIER

OPTIMIZACIÓN

Para mostrar la fecha en nuestro LOG, una de las opciones que es usar %d. Sin embargo esta forma delega en el api del JDK y en la clase SimpleDateFormat que no tiene buen rendimiento. Es mejor usar alguno de los propios formateadores de Log4J para fechas.

Cambiando nuestro código de %d a %d{ABSOLUTE} o %d{DATE} de esta forma ganaremos en rendimiento.

Otras dos opciones que son realmente lentas a la hora de grabar información al log son %M (nombre del método) y %F (nombre del fichero que generó el Log) evitémoslas dentro de lo posible.

Otra de las cosas que suele ralentizar el funcionamiento del Log es el uso que hacemos por defecto de los appenders de consola:

log4j.appender.miappender=org.apache.log4j.ConsoleAppender

Este tipo de appenders en un primer momento nos parecen muy útiles por ser muy sencillos, sin embargo en el entorno de producción real no son tan útiles ya que los administradores prefieren un appender a fichero FileAppender.

Lo que mucha gente no conoce es que no solo es más útil sino que también es mucho más rápido usar un FileAppender que un ConsoleAppender . Concretamente el FileAppender es el doble de rápido que el ConsoleAppender . Por lo tanto es interesante en producción desactivar los appender de Consola que tengamos.

El FileAppender como su nombre indica escribe en un fichero. Ahora bien la forma de escribir es continua es decir cada vez que llega un mensaje lo escribe en el fichero lo cual en un principio parece razonable pero implica también que estamos escribiendo continuamente. Esto se debe a que el FileAppender tiene configurada la siguiente opción por defecto.

log4j.appender.FILE.ImmediateFlush=true

Si cambiamos esta opción por false ganaremos rendimiento ya que el FileAppender usará un Buffer para cachear un conjunto de mensajes antes de escribirlos al disco. Simplemente activando esta opción el rendimiento del log puede mejorar un 50% o más.

USO DE: IsDebugEnabled()

Si el LOG de una aplicación está desactivado por completo o sólo para unos niveles(error, warn), el coste de una solicitud de LOG consiste en: llamada (invocación) al método + una comparación de enteros.

Supone un coste en nanosegundos, pequeño, pero si tenemos la aplicación llena de líneas de LOG, que no llaman a info, poco a poco vamos acumulando tiempo desperdiciado.

Si se pone logger.isDebugEnabled(), en el peor de los casos se pierde tiempo en evaluar la condición, pero hablamos de un tiempo inapreciable, mucho menor que en el caso de no usar la sentencia.

Para optimizar la ejecución de código al utilizar Log4J es recomendable escribir trazas de la siguiente forma:

```
If ( logger.isDebugEnabled() ) {
        Logger.debug( " Mi texto…para la vecinita " );
}
```

Antes de llamar a debug, el intérprete de Java tiene que evaluar una expresión, y esto lleva tiempo. Si no está activado el debug, evitamos esta evaluación y por tanto ahorramos tiempo. Por eso siempre que no vayamos a poner el nivel info es mejor preguntar si estamos en ese nivel.

CONSOLE APPENDER vs FILE APPENDER

Log4J tiene 2 Appenders principales: el **Console Appender** y el **File Appender**.

El ConsoleAppender imprime la información en la consola mientras que el FileAppender escribe a un fichero (normalmente configurado para que rote al llegar a un tamaño o pase el día).

Un error típico en el uso del Log4J es dejar el ConsoleAppender en Producción. Es un error porque imprimir texto a la pantalla es un gran trabajo para el Sistema Operativo que tiene que escribir las letras, hacer scroll, cambiar de línea,… de hecho en Windows escribir a la consola es una operación bloqueantes y sin buffer, por lo que escribir muchos datos a la consola empeora notablemente el rendimiento.

Con el FileAppender se obtiene una velocidad de más de 2 veces que cuando se usa el ConsoleAppender.

El FileAppender también se puede optimizar colocando el parámetro:

<param name="ImmediateFlush" value="false" />

Por defecto el FileAppender hace un flush a disco después de cada evento. Indicándole esta propiedad el StreamWriter hace un buffer en memoria antes de escribir a disco.

Otra optimización es usar el parámetro:

<param name="BufferedIO" value="true" />

Con esta opción el fichero se mantiene abierto

REFERENCIAS

http://www.arquitecturajava.com/log4j-optimizacion/

http://unpocodejava.wordpress.com/2011/05/30/configurando-los-appenders-de-log4j-para-produccion/

http://gigo.lacoctelera.net/post/2006/05/16/optimizando-velocidad-log4j

http://prietopa.wordpress.com/2011/09/25/mejorando-el-log-con-log4j-isdebugenabled/

http://logging.apache.org/log4j/1.2/manual.html#performance