

(<http://baeldung.com>)

# Una guía para la logística

Última modificación: 8 de mayo de 2018

por Eric Goebelbecker (<http://www.baeldung.com/author/eric-goebelbecker/>) (<http://www.baeldung.com/author/eric-goebelbecker/>)

**Java** (<http://www.baeldung.com/category/java/>) +

**Explotación forestal** (<http://www.baeldung.com/tag/logging/>)

Supervise y solucione problemas de aplicaciones y servicios Java con Datadog:

**>> Pruébalo gratis!** (</datadog>)

## 1. Información general

Logback (<https://logback.qos.ch/>) es uno de los marcos de registro más utilizados en la comunidad de Java. Es un reemplazo para su predecesor, Log4j. (<https://logback.qos.ch/reasons-to-switch.html>) Logback ofrece una implementación más rápida que Log4j, ofrece más opciones de configuración y más flexibilidad para archivar archivos de registro antiguos. Esta introducción presentará la arquitectura de Logback y le mostrará cómo puede usarla para mejorar sus aplicaciones.

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador  
Desarrollador Senior  
Desarrollador principal  
Arquitecto  
Gerente

## 2. Arquitectura de Logback

Tres clases comprenden la arquitectura Logback; *Logger*, *Appender* y *Layout*.

Un registrador es un contexto para mensajes de registro. Esta es la clase con la que las aplicaciones interactúan para crear mensajes de registro.

Los participantes colocan mensajes de registro en sus destinos finales. Un *Logger* puede tener más de un *Appender*. Generalmente pensamos que los Anexos están adjuntos a archivos de texto, pero Logback es mucho más potente que eso.

*Layout* prepara mensajes para la salida. Logback admite la creación de clases personalizadas para formatear mensajes, así como opciones de configuración robustas para las existentes.

## 3. Configuración

### 3.1. Dependencia de Maven

Logback usa Simple Logging Facade para Java (SLF4J) como su interfaz nativa. Antes de que podamos comenzar a registrar mensajes, necesitamos agregar Logback y Slf4j a nuestro *pom.xml*:

```

1  <dependency>
2      <groupId>ch.qos.logback</groupId>
3      <artifactId>logback-core</artifactId>
4      <version>1.2.3</version>
5  </dependency>
6
7  <dependency>
8      <groupId>org.slf4j</groupId>
9      <artifactId>slf4j-api</artifactId>
10     <version>1.8.0-beta2</version>
11     <scope>test</scope>
12 </dependency>

```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Maven Central tiene la última versión de Logback Core (<https://search.maven.org/#search%7Cgav%7C1%7Corg%3A%22ch.qos.logback%22%20AND%20a%3A%22logback-core%22>) y la versión más reciente de

*slf4j-api*

(<https://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.slf4j%22%20AND%20a%3A%22slf4j-api%22>) .

## 3.2. Classpath

Logback también requiere *logback-classic.jar*

(<https://search.maven.org/#search%7Cga%7C1%7Clogback-classic>) en classpath para runtime.

Añadiremos esto a *pom.xml* como una dependencia de prueba:

```
1 <dependency>
2   <groupId>ch.qos.logback</groupId>
3   <artifactId>logback-classic</artifactId>
4   <version>1.2.3</version>
5 </dependency>
```

## 4. Ejemplo básico y configuración

Comencemos con un ejemplo rápido del uso de Logback en una aplicación.

Primero, necesitamos un archivo de configuración. Crearemos un archivo de texto llamado *logback.xml* y lo *colocaremos* en algún lugar de nuestro classpath:

```
1 <configuration>
2   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3     <encoder>
4       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
5     </encoder>
6   </appender>
7
8   <root level="debug">
9     <appender-ref ref="STDOUT" />
10  </root>
11 </configuration>
```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

A continuación, necesitamos una clase simple con un método *principal*:

```

1 public class Example {
2
3     private static final Logger logger
4         = LoggerFactory.getLogger(Example.class);
5
6     public static void main(String[] args) {
7         logger.info("Example log from {}", Example.class.getSimpleName());
8     }
9 }

```

Esta clase crea un *registrador* y llama a *info()* para generar un mensaje de registro.

Cuando ejecutamos el *ejemplo*, vemos nuestro mensaje registrado en la consola:

```

1 20:34:22.136 [main] INFO Example - Example log from Example

```

Es fácil ver por qué Logback es tan popular; estamos corriendo en minutos.

Esta configuración y código nos dan algunas pistas sobre cómo funciona esto.

1. Tenemos un *appender* llamado *STDOUT* que hace referencia a un nombre de clase *ConsoleAppender*.
2. Hay un patrón que describe el formato de nuestro mensaje de registro.
3. Nuestro código crea un *Logger* y le pasamos nuestro mensaje a través de un método de *información()*.

Ahora que entendemos lo básico, echemos un vistazo más de cerca.

## 5. Contextos del *registrador*

Cuál de estos es el más cercano a su trabajo / función actual?

### 5.1. Creando un contexto

Para registrar un mensaje en Logback, inicializamos un *Logger* desde SLF4J o Logback:

```

1 private static final Logger logger
2     = LoggerFactory.getLogger(Example.class);

```

Y luego lo usó:

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

```
1 | logger.info("Example log from {}", Example.class.getSimpleName());
```

Este es nuestro contexto de registro. Cuando lo creamos, pasamos a *LoggerFactory* nuestra clase. Esto le da al *registrador* un nombre (también hay una sobrecarga que acepta una *cadena*).

Los contextos de registro existen en una jerarquía que se asemeja mucho a la jerarquía de objetos de Java:

1. **Un registrador es un ancestro cuando su nombre, seguido de un punto, prefija el nombre de un registrador descendiente**
2. **Un registrador es un padre cuando no hay ancestros entre él y un niño**

Por ejemplo, la clase *Ejemplo* a continuación está en el paquete *com.baeldung.logback*. Hay otra clase llamada *ExampleAppender* en el paquete *com.baeldung.logback.appenders*.

El registrador de *ExampleAppender* es un elemento secundario del *Logger* del *ejemplo*.

**Todos los registradores son descendientes del registrador de raíz predefinido.**

A *Logger* tiene un *nivel*, que se puede establecer ya sea a través de configuración o con *Logger.setLevel()*. Establecer el nivel en el código anula los archivos de configuración.

Los niveles posibles son, en orden de prioridad: *RASTREO*, *DEPURACIÓN*, *INFORMACIÓN*, *ADVERTENCIA* y *ERROR*. Cada nivel tiene un método correspondiente que usamos para registrar un mensaje en ese nivel.

**Si un registrador no tiene asignado explícitamente un nivel, hereda el nivel de su antecesor más cercano.** El registrador de raíz se predetermina a *DEPURAR*. Veremos cómo sobrescribir esto a continuación.

## 5.2. Usando un contexto

Vamos a crear un programa de ejemplo que demuestre el uso de un contexto dentro de las jerarquías de registro:

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador  
Desarrollador Senior  
Desarrollador principal  
Arquitecto  
Gerente

```

1 ch.qos.logback.classic.Logger parentLogger =
2   (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung")
3
4 parentLogger.setLevel(Level.INFO);
5
6 Logger childLogger =
7   (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung")
8
9 parentLogger.warn("This message is logged because WARN > INFO.");
10 parentLogger.debug("This message is not logged because DEBUG < INFO.");
11 childLogger.info("INFO == INFO");
12 childLogger.debug("DEBUG < INFO");

```

Cuando ejecutamos esto, vemos estos mensajes:

```

1 20:31:29.586 [main] WARN com.baeldung.logback - This message is logged
2 20:31:29.594 [main] INFO com.baeldung.logback.tests - INFO == INFO

```

Comenzamos recuperando un *Logger* llamado *com.baeldung.logback* y lo lanzamos a *ch.qos.logback.classic.Logger*.

Se necesita un contexto de Logback para establecer el nivel en la siguiente declaración; tenga en cuenta que el registrador abstracto SLF4J no implementa *setLevel()*.

Establecemos el nivel de nuestro contexto en *INFO*; luego creamos otro registrador llamado *com.baeldung.logback.tests*.

Registramos dos mensajes con cada contexto para demostrar la jerarquía. Logback registra el *WARN* e *INFO* mensajes y filtra los mensajes *DEBUG*.

Ahora, usemos el registrador de raíz:

```

1 ch.qos.logback.classic.Logger logger =
2   (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung")
3 logger.debug("Hi there!");
4
5 Logger rootLogger =
6   (ch.qos.logback.classic.Logger) LoggerFactory.getLogger(org.slf4j.Logger.ROOT_LOGGER_NAME);
7 logger.debug("This message is logged because DEBUG == DEBUG.");
8
9 rootLogger.setLevel(Level.ERROR);
10
11 logger.warn("This message is not logged because WARN < ERROR.");
12 logger.error("This is logged.");

```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Vemos estos mensajes cuando ejecutamos este fragmento:

```
1 20:44:44.241 [main] DEBUG com.baeldung.logback - Hi there!
2 20:44:44.243 [main] DEBUG com.baeldung.logback - This message is logged
3 20:44:44.243 [main] ERROR com.baeldung.logback - This is logged.
```

Para concluir, comenzamos con un contexto *Logger* e *imprimimos* un mensaje *DEBUG*.

Luego recuperamos el registrador raíz usando su nombre estáticamente definido y establecemos su nivel en *ERROR*.

Y finalmente, demostramos que Logback realmente filtra cualquier declaración menos que un error.

### 5.3. Mensajes parametrizados

A diferencia de los mensajes en los fragmentos de muestra anteriores, la mayoría de los mensajes de registro útiles requerían anexar *cadenas*. Esto implica asignar memoria, serializar objetos, concatenar *cadenas* y posiblemente limpiar la basura más tarde.

Considera el siguiente mensaje:

```
1 log.debug("Current count is " + count);
```

Se incurre en el costo de construir el mensaje ya sea que el Logger registre el mensaje o no.

Logback ofrece una alternativa con sus mensajes parametrizados:

```
1 log.debug("Current count is {}", count);
```

**Las llaves {} aceptarán cualquier *Objeto* y usarán su método *toString()* para construir un mensaje solo después de verificar que se requiere el mensaje de registro.**

Probemos algunos parámetros diferentes:

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

```

1 String message = "This is a String";
2 Integer zero = 0;
3
4 try {
5     logger.debug("Logging message: {}", message);
6     logger.debug("Going to divide {} by {}", 42, zero);
7     int result = 42 / zero;
8 } catch (Exception e) {
9     logger.error("Error dividing {} by {} ", 42, zero, e);
10 }

```

Este fragmento produce:

```

1 21:32:10.311 [main] DEBUG com.baeldung.logback.LogbackTests - Logging
2 21:32:10.316 [main] DEBUG com.baeldung.logback.LogbackTests - Going to
3 21:32:10.316 [main] ERROR com.baeldung.logback.LogbackTests - Error d
4 java.lang.ArithmeticException: / by zero
5     at com.baeldung.logback.LogbackTests.givenParameters_ValuesLogged(Lc
6     ...

```

Vemos cómo una *String*, una *int* y un *entero* pueden pasarse como parámetros.

Además, cuando se pasa una *excepción* como último argumento para un método de registro, Logback imprimirá el seguimiento de la pila por nosotros.

## 6. Configuración detallada

En los ejemplos anteriores, estábamos usando el archivo de configuración de 11 líneas que creamos en la sección 4 para imprimir los mensajes de registro en la consola. Este es el comportamiento predeterminado de Logback; si no puede encontrar un archivo de configuración, crea un *ConsoleAppender* y lo asocia con el registrador raíz.

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

### 6.1. Localización de información de configuración

Desarrollador principal

Se puede colocar un archivo de configuración en la ruta de clases y se lo denomina *logback.xml* o *logback-test.xml*.

Arquitecto

A continuación, le mostramos cómo Logback intentará encontrar datos de configuración:

Gerente



1. Busque archivos llamados *logback-test.xml* , *logback.groovy* o *logback.xml* en classpath, en ese orden.
2. Si la biblioteca no encuentra esos archivos, intentará utilizar el *ServiceLoader* (<https://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html>) de Java para localizar un implementador del *com.qos.logback.classic.spi.Configurator*.
3. Configurarse para registrar la salida directamente a la consola

Nota: la versión actual de Logback no admite la configuración de Groovy debido a que no existe una versión de Groovy compatible con Java 9.

## 6.2. Configuración básica

Echemos un vistazo más de cerca a nuestra configuración de ejemplo.

El archivo completo está en etiquetas `<configuration>` .

**Vemos una etiqueta que declara un *Appender* de tipo *ConsoleAppender* , y lo nombra *STDOUT* . Anidado dentro de esa etiqueta es un codificador. Tiene un patrón con lo que se parece a los códigos de escape de *estilo printf* :**

```

1 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
2   <encoder>
3     <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %ms
4   </encoder>
5 </appender>
```

Por último, vemos una etiqueta *raíz* . Esta etiqueta establece el registrador de raíz en el modo *DEPURAR* y asocia su resultado con el *apéndice* llamado *STDOUT* :

```

1 <root level="debug">
2   <appender-ref ref="STDOUT" />
3 </root>
```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

## 6.3. Resolución de problemas de configuración

Los archivos de configuración de Logback pueden complicarse, por lo que hay varios mecanismos integrados para solucionar problemas.

Para ver información de depuración como Logback procesa la configuración, puede activar el registro de depuración:

```
1 <configuration debug="true">
2   ...
3 </configuration>
```

Logback imprimirá información de estado en la consola mientras procesa la configuración:

```
1 23:54:23,040 |-INFO in ch.qos.logback.classic.LoggerContext[default]
2   at [file:/Users/egoebelbecker/ideaProjects/logback-guide/out/test/r
3 23:54:23,230 |-INFO in ch.qos.logback.core.joran.action.AppenderActio
4   of type [ch.qos.logback.core.ConsoleAppender]
5 23:54:23,236 |-INFO in ch.qos.logback.core.joran.action.AppenderActio
6 23:54:23,247 |-INFO in ch.qos.logback.core.joran.action.NestedComple>
7   [ch.qos.logback.classic.encoder.PatternLayoutEncoder] for [encoder_
8 23:54:23,308 |-INFO in ch.qos.logback.classic.joran.action.RootLogger
9 23:54:23,309 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAc
10 23:54:23,310 |-INFO in ch.qos.logback.classic.joran.action.Configurat
11 23:54:23,313 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator
12   as safe fallback point
```

Si se encuentran advertencias o errores al analizar el archivo de configuración, Logback escribe los mensajes de estado en la consola.

Hay un segundo mecanismo para imprimir información de estado:

```
1 <configuration>
2   <statusListener class="ch.qos.logback.core.status.OnConsoleStatusL
3   ...
4 </configuration>
```

**El *StatusListener* intercepta los mensajes de estado y los imprime durante la configuración y mientras se ejecuta el programa.**

El resultado de todos los archivos de configuración se imprime, lo que lo hace útil para ubicar archivos de configuración "desdichados" en el classpath.

## 6.4. Recargando la configuración automáticamente

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Recargar la configuración de registro mientras se ejecuta una aplicación es una poderosa herramienta de solución de problemas. Logback hace esto posible con el parámetro de *escaneo*:

```
1 <configuration scan="true">
2   ...
3 </configuration>
```

El comportamiento predeterminado es analizar el archivo de configuración para ver los cambios cada 60 segundos. Modifique este intervalo agregando *scanPeriod*:

```
1 <configuration scan="true" scanPeriod="15 seconds">
2   ...
3 </configuration>
```

Podemos especificar valores en milisegundos, segundos, minutos u horas.

## 6.5. Modificación de *madereros*

En nuestro archivo de ejemplo anterior, configuramos el nivel del registrador de raíz y lo asociamos con el *apéndice de la consola*.

Podemos establecer el nivel para cualquier registrador:

```
1 <configuration>
2   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3     <encoder>
4       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}
5     </encoder>
6   </appender>
7   <logger name="com.baeldung.logback" level="INFO" />
8   <logger name="com.baeldung.logback.tests" level="WARN" />
9   <root level="debug">
10     <appender-ref ref="STDOUT" />
11   </root>
12 </configuration>
```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Agreguemos esto a nuestro classpath y ejecutemos este código:

```

1  Logger foobar =
2      (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung.foobar");
3  Logger logger =
4      (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung.logger");
5  Logger testslogger =
6      (ch.qos.logback.classic.Logger) LoggerFactory.getLogger("com.baeldung.testslogger");
7
8  foobar.debug("This is logged from foobar");
9  logger.debug("This is not logged from logger");
10 logger.info("This is logged from logger");
11 testslogger.info("This is not logged from tests");
12 testslogger.warn("This is logged from tests");

```

Vemos esta salida:

```

1  00:29:51.787 [main] DEBUG com.baeldung.foobar - This is logged from foobar
2  00:29:51.789 [main] INFO com.baeldung.logger - This is logged from logger
3  00:29:51.789 [main] WARN com.baeldung.testslogger - This is logged from tests

```

Al no establecer el nivel de nuestros registradores programáticamente, la configuración los establece; ***com.baeldung.foobar* hereda de depuración del registrador de la raíz.**

Los madereros también heredan el *appender-ref* del registrador de raíz. Como veremos a continuación, podemos anular esto.

## 6.6. Sustitución variable

Los archivos de configuración de Logback soportan variables. Definimos variables dentro del script de configuración o externamente. Se puede especificar una variable en cualquier punto de un script de configuración en lugar de un valor.

Por ejemplo, aquí está la configuración para un *FileAppender*:

```

1  <property name="LOG_DIR" value="/var/log/appender-ref"></property>
2  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
3      <file>${LOG_DIR}/tests.log</file>
4      <append>true</append>
5      <encoder>
6          <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
7      </encoder>
8  </appender>

```

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

En la parte superior de la configuración, *declaramos* una *propiedad* denominada `LOG_DIR`. Luego lo usamos como parte de la ruta al archivo dentro de la definición del *apéndice*.

Las propiedades se declaran en una etiqueta `<property>` en los scripts de configuración. Pero también están disponibles desde fuentes externas, como las propiedades del sistema. Podríamos omitir la declaración de *propiedad* en este ejemplo y establecer el valor de `LOG_DIR` en la línea de comando:

```
1 | $ java -DLOG_DIR=/var/log/application com.baeldung.logback.LogbackTest
```

Especificamos el valor de la propiedad con `$ {propertyname}`. Logback implementa variables como reemplazo de texto. La sustitución de variables puede ocurrir en cualquier punto de un archivo de configuración donde se puede especificar un valor.

## 7. Anexos

Los madereros pasan *LoggingEvents* a los *Appenders*. Los *appenders* hacen el trabajo real de iniciar sesión. Normalmente pensamos en iniciar sesión como algo que va a un archivo o la consola, pero Logback es capaz de mucho más. *Logback-core* proporciona varios apéndices útiles.

### 7.1. ConsoleAppender

Ya hemos visto a *ConsoleAppender* en acción. A pesar de su nombre, *ConsoleAppender* agrega mensajes a `System.out`.

Utiliza un *OutputStreamWriter* para almacenar en búfer la E / S, por lo que dirigirlo a `System.err` no da como resultado la escritura sin búfer.

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

### 7.2. FileAppender

*FileAppender* agrega mensajes a un archivo. Es compatible con una amplia gama de parámetros de configuración. Agreguemos *FileAppender* a nuestra configuración básica:

```

1 <configuration debug="true">
2   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3     <!-- encoders are assigned the type
4         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
5     <encoder>
6       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} %n</pattern>
7     </encoder>
8   </appender>
9
10  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
11    <file>tests.log</file>
12    <append>true</append>
13    <encoder>
14      <pattern>%-4relative [%thread] %-5level %logger{35} - %ms</pattern>
15    </encoder>
16  </appender>
17
18  <logger name="com.baeldung.logback" level="INFO" />
19  <logger name="com.baeldung.logback.tests" level="WARN">
20    <appender-ref ref="FILE" />
21  </logger>
22
23  <root level="debug">
24    <appender-ref ref="STDOUT" />
25  </root>
26 </configuration>

```

El *FileAppender* está configurado con un nombre de archivo a través de `<archivo>`. La etiqueta `<append>` indica al *Appender* que se agregue a un archivo existente en lugar de truncarlo. Si ejecutamos la prueba varias veces, vemos que la salida de registro se agrega al mismo archivo.

Si volvemos a ejecutar nuestra prueba desde arriba, los mensajes de *com.baeldung.logback.tests* van a la consola y a un archivo llamado *tests.log*.

**El registrador descendiente hereda la asociación del registrador de raíz con *ConsoleAppender*. ¿Cómo se desasocia el *FileAppender*. Los appenders son acumulativos o no a su trabajo / función actual?**

Podemos anular este comportamiento:

```

1 <logger name="com.baeldung.logback.tests" level="WARN" additivity="false">
2   <appender-ref ref="FILE" />
3 </logger>
4
5 <root level="debug">
6   <appender-ref ref="STDOUT" />
7 </root>

```

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Establecer *aditividad en falso* deshabilita el comportamiento predeterminado. *Las pruebas* no se registrarán en la consola, ni tampoco ninguno de sus descendientes.

## 7.3. RollingFileAppender

A menudo, anexar mensajes de registro al mismo archivo no es el comportamiento que necesitamos. Queremos que los archivos se "transfieran" en función del tiempo, el tamaño del archivo de registro o una combinación de ambos.

Para esto, tenemos *RollingFileAppender*:

```

1  <property name="LOG_FILE" value="LogFile" />
2  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
3      <file>${LOG_FILE}.log</file>
4      <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
5          <!-- daily rollover -->
6          <fileNamePattern>${LOG_FILE}-%d{yyyy-MM-dd}.gz</fileNamePattern>
7
8          <!-- keep 30 days' worth of history capped at 3GB total size -->
9          <maxHistory>30</maxHistory>
10         <totalSizeCap>3GB</totalSizeCap>
11     </rollingPolicy>
12     <encoder>
13         <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
14     </encoder>
15 </appender>

```

Un *RollingFileAppender* tiene una *RollingPolicy*. En esta configuración de muestra, vemos una *TimeBasedRollingPolicy*.

Similar a *FileAppender*, configuramos este *appender* con un nombre de archivo. Declaramos una propiedad y la usamos para este porque reutilizaremos el nombre del archivo a continuación.

Definimos un *fileNamePattern* dentro de *RollingPolicy*. Este patrón define no solo el nombre de los archivos, sino también la frecuencia con la que se deben rodar. *TimeBasedRollingPolicy* examina el patrón y reemplaza el período más definido.

Por ejemplo:

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente



```

1 <property name="LOG_FILE" value="LogFile" />
2 <property name="LOG_DIR" value="/var/logs/application" />
3 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender" >
4     <file>${LOG_DIR}/${LOG_FILE}.log</file>
5     <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy" >
6         <fileNamePattern>${LOG_DIR}/%d{yyyy/MM}/${LOG_FILE}.gz</fileNamePattern>
7         <totalSizeCap>3GB</totalSizeCap>
8     </rollingPolicy>

```

El archivo de registro activo es `/var/logs/application/LogFile`. Este archivo se transfiere al comienzo de cada mes en `/Año actual/Mes actual/LogFile.gz` y *RollingFileAppender* crea un nuevo archivo activo.

Cuando el tamaño total de los archivos archivados llega a 3 GB, *RollingFileAppender* borra los archivos archivados primero en *entrar* - primero en salir.

Hay códigos para una semana, una hora, un minuto, un segundo e incluso milisegundos. Logback tiene una referencia aquí (<https://logback.qos.ch/manual/appenders.html#TimeBasedRollingPolicy>)

*RollingFileAppender* también tiene soporte integrado para comprimir archivos. Comprime nuestros archivos enrollados porque los ha llamado *LogFile.gz*.

*TimeBasedPolicy* no es nuestra única opción para *transferir* archivos. Logback también ofrece *SizeAndTimeBasedRollingPolicy*, que se extenderá en función del tamaño del archivo de registro actual y del tiempo. También ofrece una *FixedWindowRollingPolicy* que lanza los nombres de los archivos de registro cada vez que se inicia el registrador.

También podemos escribir nuestra propia *RollingPolicy* (<https://logback.qos.ch/manual/appenders.html#onRollingPolicies>).

## 7.4. Anexos personalizados

Podemos crear apéndices personalizados extendiendo una de las clases de *appenders* base de Logback. Tenemos un tutorial para crear apéndices personalizados aquí (<http://www.baeldung.com/custom-logback-appender>).

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

## 8. Diseños



*Formato de mensajes de registro de formato.* Al igual que el resto de Logback, los *diseños* son extensibles y podemos crear los nuestros. (<https://logback.qos.ch/manual/layouts.html#writingYourOwnLayout>) Sin embargo, *PatternLayout* predeterminado ofrece lo que la mayoría de las aplicaciones necesitan y más.

Hemos utilizado *PatternLayout* en todos nuestros ejemplos hasta el momento:

```
1 <encoder>
2   <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
3 </encoder>
```

Este script de configuración contiene la configuración de *PatternLayoutEncoder*. Pasamos un *Encoder* a nuestro *Appender*, y este codificador usa *PatternLayout* para formatear los mensajes.

El texto en la etiqueta *<pattern>* define cómo se están formateando los mensajes de registro. ***PatternLayout* implementa una gran variedad de palabras de conversión y modificadores de formato para crear patrones.**

Vamos a romper esto. *PatternLayout* reconoce las palabras de conversión con un%, por lo que las conversiones en nuestro patrón generan:

- *%d{HH:mm:ss.SSS}* - una marca de tiempo con horas, minutos, segundos y milisegundos
- *[%thread]* - el nombre del hilo que genera el mensaje de registro, rodeado por corchetes
- *%-5level* - el nivel del evento de registro, rellenado a 5 caracteres
- *%logger{36}* - el nombre del registrador, truncado a 35 caracteres
- *%msg%n*: los mensajes de registro seguidos por el carácter separador de líneas dependiente de la plataforma

Entonces vemos mensajes similares a esto:

```
1 21:32:10.311 [main] DEBUG com.baeldung.logback.LogbackTests - Logging
```

Aquí (<https://logback.qos.ch/manual/layouts.html#conversionWord>) puede encontrar una lista exhaustiva de palabras de conversión y modificadores de formato.

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

## 9. Conclusión

En esta extensa guía, cubrimos los fundamentos del uso de Logback en una aplicación.

Analizamos los tres componentes principales en la arquitectura de Logback: registradores, anexos y diseño. Logback tiene potentes scripts de configuración, que usamos para manipular componentes para filtrar y formatear mensajes. También examinamos los dos apéndices de archivos más utilizados para crear, transferir, organizar y comprimir archivos de registro.

Como de costumbre, los fragmentos de código se pueden encontrar en GitHub (<https://github.com/eugenp/tutorials/tree/master/logging-modules/logback>).

## Supervise y solucione problemas de aplicaciones y servicios Java con Datadog:

>> Pruébalo gratis! (/datadog)

Deja una respuesta

¡Sé el primero en comentar!



Start the discussion

☒ Suscribirse ▼

**¿Cuál de estos es el más cercano a su trabajo / función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

## CATEGORÍAS

PRIMAVERA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/](http://www.baeldung.com/category/spring/))  
DESCANSO ([HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/](http://www.baeldung.com/category/rest/))  
JAVA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/](http://www.baeldung.com/category/java/))  
SEGURIDAD ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](http://www.baeldung.com/category/security-2/))  
PERSISTENCIA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](http://www.baeldung.com/category/persistence/))  
JACKSON ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/](http://www.baeldung.com/category/jackson/))  
HTTPCLIENT ([HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/](http://www.baeldung.com/category/http/))  
KOTLIN ([HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](http://www.baeldung.com/category/kotlin/))

## SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA ([HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL](http://www.baeldung.com/java-tutorial))  
JACKSON JSON TUTORIAL ([HTTP://WWW.BAELDUNG.COM/JACKSON](http://www.baeldung.com/jackson))  
TUTORIAL DE HTTPCLIENT 4 ([HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE](http://www.baeldung.com/httpclient-guide))  
REST CON SPRING TUTORIAL ([HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/](http://www.baeldung.com/rest-with-spring-series/))  
TUTORIAL DE SPRING PERSISTENCE ([HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/](http://www.baeldung.com/persistence-with-spring-series/))  
SEGURIDAD CON SPRING ([HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING](http://www.baeldung.com/security-spring))

## ACERCA DE

ACERCA DE BAELDUNG ([HTTP://WWW.BAELDUNG.COM/ABOUT/](http://www.baeldung.com/about/))  
LOS CURSOS ([HTTP://COURSES.BAELDUNG.COM/](http://courses.baeldung.com/))  
TRABAJO DE CONSULTORÍA ([HTTP://WWW.BAELDUNG.COM/CONSULTING](http://www.baeldung.com/consulting))  
META BAELDUNG ([HTTP://META.BAELDUNG.COM/](http://meta.baeldung.com/))  
EL ARCHIVO COMPLETO ([HTTP://WWW.BAELDUNG.COM/FULL-ARCHIVE](http://www.baeldung.com/full-archive))  
ESCRIBIR PARA BAELDUNG ([HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES](http://www.baeldung.com/contribution-guidelines))  
CONTACTO ([HTTP://WWW.BAELDUNG.COM/CONTACT](http://www.baeldung.com/contact))  
INFORMACIÓN DE LA COMPAÑÍA ([HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO](http://www.baeldung.com/baeldung-company-info))  
TÉRMINOS DE SERVICIO ([HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE](http://www.baeldung.com/terms-of-service))  
POLÍTICA DE PRIVACIDAD ([HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY](http://www.baeldung.com/privacy-policy))  
EDITORES ([HTTP://WWW.BAELDUNG.COM/EDITORS](http://www.baeldung.com/editors))  
KIT DE MEDIOS (PDF) ([HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF](https://s3.amazonaws.com/baeldung.com/baeldung+-+media+kit.pdf))

**¿Cuál de estos es el más  
cercano a su trabajo /  
función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

**¿Cuál de estos es el más  
cercano a su trabajo /  
función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente