Unidad 6. Técnicas de programación segura.Anexo I - Log4Java2 - Gestor de logs

Descargar estos apuntes

Índice

Anexo I - Log4Java2 - Gestor de logs

Registros o logs

El registro es el proceso de escribir mensajes de registro en cualquier archivo, base de datos, consola, etc. para mantener un registro de eventos que ocurren en un sistema. El registro es una parte esencial de cualquier aplicación para depurar y monitorear el sistema.

El software con suficiente registro y monitorización le permitirá detectar posibles incidentes cuando su código se despliegue en un entorno de producción. `Log4j` se utiliza para gestionar el registro de información en una aplicación.

Además, en términos de seguridad, el registro es una parte esencial de cualquier aplicación. Si su aplicación se ve comprometida, el registro le ayudará a rastrear el origen de un ataque, accesos indebidos, accesos no autorizados y a tomar medidas para evitar que vuelva a ocurrir.

Hasta ahora hemos usado la instrucción SOP `System.out.println()` para imprimir mensaje de registro. Este sistema tiene algunas desventajas:

- Podemos imprimir mensajes de registro solo en la consola. Por lo tanto, cuando se cierre la consola, perderemos -dos los registros.
- No podemos almacenar mensajes de registro en ningún lugar permanente. Estos mensajes se imprimirán uno por uno en la consola porque es un entorno de un solo hilo.
- No podemos configurar diferentes niveles de registro como INFO, DEBUG, ERROR, etc.
- No podemos configurar el formato de registro, como la fecha, la hora, el nombre de la clase, el nombre del método y que se aplique a todos los mensajes de registro.
- No podemos configurar el destino de registro, como la consola, el archivo, la base de datos, etc.

Para solventar estos problemas, se utiliza el framework Log4j. Log4j es un framework de código abierto proporcionado por Apache solo para proyectos de Java.

> **Versiones Log4j2**
>
> Log4j es un sistema de registro donde la API (llamada Log4j API) y su implementación (llamada Log4j Core) están separadas claramente la una de la otra.
>
> Esto permite que la API de Log4j proporcione una interfaz que sea fácil de usar de una manera correcta y a prueba de futuro. Consulta las páginas de API de Java, API de Kotlin y API de Scala para obtener más información.

Para usar Log4j2 en nuestro código solo necesitamos agregar las siguientes librerías / dependencias en nuestro proyecto:

```
log4j-api-<version>.jar
log4j-core-<version>.jar
```

Configuración automática de Log4j2

Podemos configurar Log4j2 con nuestra aplicación utilizando un archivo de configuración escrito en formato XML, JSON, YAML o propiedades. También podemos hacerlo mediante código pero, por ahora, nos vamos a centrar en la configuración utilizando archivos de configuración.

Log4j tiene la capacidad de configurarse automáticamente durante la inicialización. Tiene un orden para buscar el archivo de configuración en la aplicación. Log4j proporcionará una configuración predeterminada si no puede localizar un archivo de configuración.

Niveles de registro

Los niveles de registro son un mecanismo para categorizar los registros. Los niveles se utilizan para identificar la gravedad de un evento. Podemos configurar fácilmente los niveles para especificar qué detalles de registro queremos ver. Log4j proporciona los siguientes niveles:

1. ALL: Para registrar todos los eventos.
2. TRACE: Un mensaje de depuración detallado, que captura típicamente el flujo a través de la aplicación.
3. DEBUG: Un evento de depuración general.
4. INFO: Un evento con fines informativos.
5. WARN: Un evento que podría posiblemente llevar a un error.
6. ERROR: Un error en la aplicación, posiblemente recuperable.
7. FATAL: Un error grave que impedirá que la aplicación continúe.
8. OFF: No se registrarán eventos.

Log4j sigue el siguiente orden:

> ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL

Si especificamos el nivel de registro como INFO, se registrarán todos los eventos INFO, WARN, ERROR y FATAL. Si especificamos el nivel de registro como WARN, se registrarán todos los eventos WARN, ERROR y FATAL. En términos simples, se considerarán todos los niveles por debajo del nivel especificado, incluido el nivel especificado.

Ejemplo de uso de Log4j2

```java
package psp.examples.u06.log4j2;

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

public class ModuleA {

    private static final Logger logger = LogManager.getLogger();

    // Log messages
    public static void main(String[] args) {
        logger.debug("It is a debug logger.");
        logger.error("It is an error logger.");
        logger.fatal("It is a fatal logger.");
        logger.info("It is a info logger.");
        logger.trace("It is a trace logger.");
        logger.warn("It is a warn logger.");
    }
}
```

La salida obtenida será la siguiente:

```
11:14:47.469 [main] ERROR psp.examples.u06.log4j2.ModuleA - It is an error logger.
11:14:47.471 [main] FATAL psp.examples.u06.log4j2.ModuleA - It is a fatal logger.
```

Aunque hemos usado todos los niveles, en la consola solo vemos dos niveles. En realidad, cuando no proporcionamos ningún archivo de configuración, por defecto Log4j utiliza una configuración predeterminada. La configuración predeterminada, proporcionada en la clase DefaultConfiguration, configurará:

- Un `ConsoleAppender` para el logger por defecto, es decir, los registros se imprimirán en la consola.
- Un `PatternLayout` configurado con el patrón:
  " %d{HH:mm:ss.SSS} [%t] %-5level %logger{36} – %msg%n " asociado al ConsoleAppender

> **Nivel de registro por defecto**
>
> Por defecto, Log4j asigna el logger raíz al nivel Level.ERROR y esos logs se imprimirán en la consola estándar.

> **Patrón por defecto**
>
> Vamos a entender el formato de patrón en el que se imprimen los logs. Dado que no hemos pasado ningún archivo de configuración, utiliza el formato por defecto que se muestra a continuación.
>
> ```
> " %d{HH:mm:ss.SSS} [%t] %-5level %logger{36} – %msg%n "
> ```

- `%d{HH:mm:ss.SSS}` is execution timestamp i.e. 18:07:15.984
- `[%t]` is thread name i.e. [main]
- `%-5level` is level name i.e. ERROR
- `%logger{36}` is logger name which we are creating as first step i.e. psp.examples.u06.log4j2
- `%msg%n` is message i.e. “ It is an error logger ” followed by a new line character.

Configuración de Log4j2

Para configurar Log4j2, necesitamos un archivo de configuración. Log4j2 admite la configuración en formato XML, JSON, YAML y propiedades.

Log4j2 tiene una arquitectura de registro bastante compleja, sin embargo la mayoría de los usuarios solo requieren estos elementos:

- **Loggers**: Son el punto de entrada del pipeline de registro, que se utiliza directamente en el código. Su configuración debe especificar qué nivel de mensajes registran y a qué `*appenders*` envían los mensajes. Los cubriremos mientras configuramos los `*loggers*`.
- **Appenders**: Son el punto de salida del pipeline de registro. Deciden a qué recurso (consola, archivo, base de datos o similar) se envía el evento de registro. Los más comunes son el `*console appender*` y el `*file appender*`.
- **Layouts**: Indican a los `*appenders*` cómo formatear el evento de registro: texto, JSON, XML o similar. Los más comunes son `*Pattern Layout*` y `*JSON Template Layout*`.

A continuación se muestra un ejemplo de configuración en formato XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

En este archivo de configuración, hemos configurado un `ConsoleAppender` que imprimirá los mensajes de registro en la consola. Hemos configurado el nivel de registro raíz como `error`, por lo que solo se imprimirán los mensajes de registro de nivel `error` o superior.

Log4j2 buscará los archivos de configuración en el siguiente orden:

1. Primero comprobará si se ha especificado un archivo de configuración en la propiedad del sistema `log4j.configurationFile`. Si no se ha especificado, buscará los archivos de configuración en el siguiente

orden:

2. Si no encuentra la propiedad del sistema buscará los archivos de configuración en el `classpath` en el siguiente orden:
   - `log4j2-test.xml`
   - `log4j2-test.json`
   - `log4j2-test.yaml`
   - `log4j2-test.yml`
   - `log4j2-test.properties`
   - `log4j2.xml`
   - `log4j2.json`
   - `log4j2.yaml`
   - `log4j2.yml`
   - `log4j2.properties`

Se puede ampliar la información en la documentación oficial de Log4j2.

Appenders

> Documentación Appenders

Podemos especificar destinos para mantener los registros de eventos. Podemos querer imprimir esos registros en la consola o en cualquier archivo externo. Los `appenders` suelen ser responsables de escribir los datos de eventos en el destino objetivo. Podemos usar varios *appenders* en una misma configuración.

Los `appenders` más comunes son:

- **ConsoleAppender**: Imprime los mensajes de registro en la consola.
- **FileAppender**: Imprime los mensajes de registro en un archivo.
- **RollingFileAppender**: Imprime los mensajes de registro en un archivo y crea un nuevo archivo cuando el tamaño del -chivo alcanza un límite.
- **DBAppender**: Imprime los mensajes de registro en una base de datos.
- **SMTPAppender**: Envía los mensajes de registro por correo electrónico.
- **SocketAppender**: Envía los mensajes de registro a un servidor remoto.
- **SyslogAppender**: Envía los mensajes de registro a un servidor Syslog.
- **JMSAppender**: Envía los mensajes de registro a un servidor JMS.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="https://logging.apache.org/xml/ns"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="
                   https://logging.apache.org/xml/ns
                   https://logging.apache.org/xml/ns/log4j-config-2.xsd">
  <Appenders>
    <Console name="CONSOLE">
      <PatternLayout pattern="%p - %m%n"/>
    </Console>
    <File name="MAIN" fileName="logs/main.log">
      <JsonTemplateLayout/>
    </File>
    <File name="DEBUG_LOG" fileName="logs/debug.log">
      <PatternLayout pattern="%d [%t] %p %c - %m%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="INFO">
      <AppenderRef ref="CONSOLE" level="WARN"/>
      <AppenderRef ref="MAIN"/>
    </Root>
    <Logger name="org.example" level="DEBUG">
      <AppenderRef ref="DEBUG_LOG"/>
    </Logger>
  </Loggers>
</Configuration>
```

En el ejemplo anterior se han configurado tres `appenders`:

- Un `ConsoleAppender` llamado CONSOLE con un `PatternLayout`.
  En este caso la salida se mostrará en la consola y se imprimirá el nivel de log y el mensaje. Por ejemplo, `INFO - It is an info logger`.
- Un `FileAppender` llamado MAIN con un `JsonTemplateLayout`.
  En este caso la salida se escribirá en el archivo `logs/main.log` en formato JSON. Por ejemplo, `{"level":"INFO","message":"It is an info logger"}`.
- Un `FileAppender` llamado DEBUG_LOG con un `PatternLayout`.
  En este caso la salida se escribirá en el archivo `logs/debug.log` con el formato de fecha, hilo, nivel de log, clase y mensaje. Por ejemplo,
  `2021-09-15 11:14:47,471 [main] ERROR psp.examples.u06.log4j2.ModuleA - It is an error logger.`

Loggers

> Documentación Loggers

Los `loggers` se utilizan directamente en el código para registrar mensajes.

Los `loggers` se configuran en el archivo de configuración de Log4j2. y se pueden configurar para enviar mensajes de registro a uno o varios `appenders`.

En el ejemplo anterior, se han configurado dos `loggers`:

- Un `logger` raíz con nivel de log INFO y dos `appenders` asociados: CONSOLE y MAIN.
  En este caso, los mensajes de log con nivel INFO o superior se enviarán al `appender` MAIN, es decir, al archivo `logs/main.log`.
  Los mensajes de log con nivel WARN o superior se enviarán también al `appender` CONSOLE, es decir, a la consola.
- Un `logger` con nombre `org.example` con nivel de log DEBUG y un `appender` asociado: DEBUG_LOG.
  En este caso, los mensajes de log con nivel DEBUG o superior se enviarán al `appender` DEBUG_LOG.

Un mismo mensaje, diferentes salidas

Fíjate en la siguiente tabla con los mensajes que se enviarían a cada `appender` en función del nivel de log, del `logger` que lo genere y de los `appenders`.

Logger name	Log event level	Appenders
org.example.foo	WARN	CONSOLE, MAIN, DEBUG_LOG
org.example.foo	DEBUG	MAIN, DEBUG_LOG
org.example.foo	TRACE	none
com.example	WARN	CONSOLE, MAIN
com.example	INFO	MAIN
com.example	DEBUG	none

Si un logger no tiene un nivel de log configurado se heredará el nivel de log del logger padre. Si no se ha configurado ningún nivel de log, se heredará el nivel de log del logger raíz.

Layouts

> Documentación Layouts

Los `layouts` se utilizan para dar formato a los mensajes de log. Log4j2 proporciona varios `layouts` predefinidos que se pueden utilizar para dar formato a los mensajes de log.

Los `layouts` más comunes son:

- **PatternLayout**: Es el `layout` más comúnmente utilizado. Permite configurar el formato de los mensajes de log utilizando un patrón.
- **JsonTemplateLayout**: Es un `layout` que genera mensajes de log en formato JSON.
- **YamlLayout**: Es un `layout` que genera mensajes de log en formato YAML.
- **HtmlLayout**: Es un `layout` que genera mensajes de log en formato HTML.

- **CsvLogLayout**: Es un `layout` que genera mensajes de log en formato CSV.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

En este archivo de configuración, hemos configurado un `ConsoleAppender` que imprimirá los mensajes de registro en la consola. Hemos configurado el nivel de registro raíz como `error`, por lo que solo se imprimirán los mensajes de registro de nivel `error` o superior.

Log4j2 buscará los archivos de configuración en el siguiente orden:

Un ejemplo de configuración de Log4j2 con un `JsonTemplateLayout` sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <JsonTemplateLayout eventTemplateUri="classpath:log4j2/templates/JsonEventLayout.json"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

Donde un posible contenido para el archivo `JsonEventLayout.json` podría ser el siguiente:

```
{
  "timeMillis": "${timeMillis}",
  "thread": "${thread}",
  "level": "${level}",
  "loggerName": "${loggerName}",
  "message": "${message}",
  "thrown": "${thrown}"
}
```

Actividades

Puedes probar a realizar las siguientes actividades para practicar con Log4j2:

1. Escribe un programa simple utilizando Log4j2 que imprima TODOS los niveles de error en la consola.
2. Utilizando el programa del ejercicio 1, genera un archivo de configuración manual para imprimir los errores FATALES en SYSTEM_ERR y el resto de los niveles de error en SYSTEM_OUT. (Añade comentarios en el XML con `<!--Tu comentario-->`)
3. Utilizando el programa del ejercicio 1, genera un archivo de configuración manual en el que los errores FATALES vayan a un archivo de texto, los errores de nivel ERROR aparezcan en rojo en la consola (SYSTEM_ERR) y el resto aparezcan en la consola normal (SYSTEM_OUT).
4. Utilizando el archivo anterior (ejercicio 3) cambia el Patrón de Salida para que lo primero que aparezca en la línea de LOG sean tus iniciales.

> **Guardar los registros en una base de datos**
>
> Log4j2 también permite guardar los registros en una base de datos.
> Investiga cómo se puede hacer y realiza un ejemplo de cómo guardar los registros en una base de datos MySQL.

Bibliografía

- Log4j2 - Documentación oficial
- Tutorial de 7 partes de MakeSeleniumEasy sobre Log4j2
- Log4j2 - Tutorialspoint