|  |
| --- |
| **TECNICATURA UNIVERSITARIA EN PROGRAMACION**  **Laboratorio II - Ciclo lectivo 2024** |

|  |  |
| --- | --- |
| **Características del Práctico : Log4J** | |
| **Unidad** | **4** |
| **Tema** | **Edición y Depuración de Programas.** |
| **Resultados de Aprendizajes** | RA1: Aplica la herramienta Log4J para obtener la bitácora de ejecución de una aplicación Java. |
| **Objetivo** | El objetivo de este trabajo práctico es que los estudiantes adquieran un entendimiento profundo de la principal herramienta propuesta para logging. |
| **Requisitos técnicos** | Utilizar Spring Boot y Gradle como sistema de gestión de dependencias.  Codificar una solución básica en Java. |
| **Fecha de inicio** | Junio de 2024 - |
| **Fecha de entrega:** | Junio de 2024 |
| **Modalidad de entrega :** | Adjuntar el trabajo en formato Word al link correspondiente en el aula virtual. |
| **Comisiones** | **Comisiones: Mañana y tarde** |
| **Modalidad del Trabajo Práctico** | Desarrollo Grupal – Entrega en aula virtual |
|  | |
| **Conclusiones:**  Los estudiantes deben resumir lo que aprendieron durante la realización del trabajo práctico y destacar cómo las herramientas de logging permiten el seguimiento de la ejecución de una aplicación. | |
| Forma de Presentación | El trabajo debe presentarse en el link correspondiente al trabajo práctico alojado en el aula virtual. |
| Tipo de Evaluación | Formativa - Grupal |
| **Modalidad de Evaluación** | |  |  |  |  | | --- | --- | --- | --- | |  |  | **>70%** | **<50 %** | | **Configuración del entorno** |  |  |  | | **Configuración del archivo de propiedades** |  |  |  | | **Codificación Java** |  |  |  | | **RESULTADOS** | **Excelente** | **Satisfactorio** | **No Satisfactorio** |   **Se establece una sumatoria de cada ítem y se determina el porcentaje individual de cada uno. Posteriormente se establece la sumatoria de todos los ítems y se lo divide por la cantidad de los mismos para determinar en cuál de los valores de la escala se encuadra.**   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |
| Bibliografía | Listas de reproducción de videos de la cátedra y material teórico del aula virtual |
| Profesores responsables del TP. | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Apellido y nombre | | Apellido y nombre | | | | | Julio Monetti | | Claudia Naveda | | | | |  |  | |  |  |  | |

DESARROLLO

**PARTE A**

Con el proyecto facilitado

**Ejercicio** **1**

Investigue y explique con sus palabras que produce la etiqueta **Loggers** en el archivo de configuración de log4j.

***En Log4j, la etiqueta <Loggers> en el archivo de configuración se utiliza para configurar los registradores (loggers) que se encargan de enviar los mensajes de registro (logs) a los destinos apropiados.***

**Ejercicio** **2**

Investigue y explique con sus palabras que produce la etiqueta **Appenders** en el archivo de configuración de log4j.

***En Log4j, la etiqueta <Appenders> en el archivo de configuración (por ejemplo, log4j2.xml) se utiliza para definir y configurar los destinos de los logs, conocidos como appenders. Los appenders son componentes que determinan dónde se enviarán los mensajes de registro (logs) generados por la aplicación.***

**Ejercicio** **3**

Investigue y explique con sus palabras que produce la etiqueta **Root Level** en el archivo de configuración de log4j.

***En Log4j, la etiqueta <Root> en el archivo de configuración se utiliza para definir el nivel de registro raíz y el appender raíz para la aplicación.***

***El nivel de registro raíz especifica el nivel mínimo de importancia de los mensajes de registro que se enviarán al appender raíz. Cualquier mensaje de registro con un nivel igual o superior al nivel de registro raíz será procesado y enviado al appender raíz para su posterior manejo.***

**Ejercicio 4**

Investigue y documente: qué es Apache Flume ?

***.Apache Flume es un sistema de recolección de datos distribuido y confiable, diseñado para mover grandes volúmenes de datos de forma eficiente desde múltiples fuentes hacia destinos de almacenamiento centralizados. Es un proyecto de código abierto de la Apache Software Foundation y se utiliza comúnmente en entornos de Big Data para la ingestión de datos***

**PARTE B**

**Ejercicio**

1. Cree una aplicación Java llamada **milog**. La misma debe contar solamente con la clase principal, la cual expone un mensaje sobre la consola informando que la aplicación se ha iniciado.

***public class Milog {***

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

***System.out.println("La aplicación milog se ha iniciado.");***

***}***

***}***

**Ejercicio**

Cree una aplicación que utilice un nivel de logging personalizado, denominado **miLogLevel**.

***import org.apache.logging.log4j.Level;***

***public class MiLogLevel extends Level {***

***private static final long serialVersionUID = 1L;***

***public static final Level MI\_LOG\_LEVEL = new MiLogLevel(35000, "MI\_LOG\_LEVEL", 7);***

***protected MiLogLevel(int level, String name, int syslogEquivalent) {***

***super(level, name, syslogEquivalent);***

***}***

***}***

Indique como crear el nuevo nivel de logging desde el archivo de configuración.

***<?xml version="1.0" encoding="UTF-8"?>***

***<Configuration status="INFO">***

***<Appenders>***

***<!-- Definir appenders aquí -->***

***</Appenders>***

***<Loggers>***

***<Root level="DEBUG">***

***<AppenderRef ref="CONSOLE"/>***

***<AppenderRef ref="FILE"/>***

***</Root>***

***<!-- Configuración para nuestro nuevo nivel de logging personalizado -->***

***<Logger name="com.example" level="miLogLevel" additivity="false">***

***<AppenderRef ref="CUSTOM\_APPENDER"/>***

***</Logger>***

***</Loggers>***

***</Configuration>***

**Ejercicio**

Cree una aplicación que envíe mensajes de logging a:

1. Consola
2. Un fichero denominado **log.txt**.
3. Investigue y aplique el logging sobre http.

a)

***import java.util.logging.Level;***

***import java.util.logging.Logger;***

***public class Main {***

***// Obtener una instancia de Logger para la clase actual***

***private static final Logger logger = Logger.getLogger(Main.class.getName());***

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

***// Enviar un mensaje de logging***

***logMessage("Este es un mensaje de prueba.");***

***}***

***// Método para enviar mensajes de logging***

***private static void logMessage(String message) {***

***// Loggear el mensaje con nivel INFO***

***logger.log(Level.INFO, message);***

***}***

***}***

b)

***import java.io.IOException;***

***import java.util.logging.FileHandler;***

***import java.util.logging.Level;***

***import java.util.logging.Logger;***

***import java.util.logging.SimpleFormatter;***

***public class Main {***

***// Obtener una instancia de Logger para la clase actual***

***private static final Logger logger = Logger.getLogger(Main.class.getName());***

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

***try {***

***// Crear un manejador de archivos para redirigir el logging a un archivo llamado "log.txt"***

***FileHandler fileHandler = new FileHandler("log.txt");***

***// Formatear los mensajes de logging de manera sencilla***

***fileHandler.setFormatter(new SimpleFormatter());***

***// Agregar el manejador de archivos al Logger***

***logger.addHandler(fileHandler);***

***} catch (IOException e) {***

***// Manejar cualquier excepción que pueda ocurrir al crear el manejador de archivos***

***e.printStackTrace();***

***}***

***// Enviar un mensaje de logging***

***logMessage("Este es un mensaje de prueba.");***

***}***

***// Método para enviar mensajes de logging***

***private static void logMessage(String message) {***

***// Loggear el mensaje con nivel INFO***

***logger.log(Level.INFO, message);***

***}***

***}***

c)

*Para agregar logging a un servidor HTTP en Java, se puede utilizar los registros (logs) de la API de java.util.logging junto con el manejo de solicitudes y respuestas HTTP proporcionado por un servidor como Jetty o Tomcat. A continuación, se mostrará cómo hacerlo con Jetty, uno de los servidores web más populares en Java:*

*1)Primero, tener Jetty agregado a tus dependencias. Si se están utilizando Maven, puedes agregarlo al archivo pom.xml de la siguiente manera:*

**<dependency>**

**<groupId>org.eclipse.jetty</groupId>**

**<artifactId>jetty-server</artifactId>**

**<version>11.0.7</version> <!-- La versión más reciente al momento de escribir esto -->**

**</dependency>**

2) *Luego, se puede configurar Jetty para que utilice un manejador de logs personalizado que envíe los mensajes de log a un archivo. Aquí hay un ejemplo de cómo hacerlo:*

***import org.eclipse.jetty.server.Server;***

***import org.eclipse.jetty.server.handler.AbstractHandler;***

***import org.eclipse.jetty.server.Request;***

***import javax.servlet.ServletException;***

***import javax.servlet.http.HttpServletRequest;***

***import javax.servlet.http.HttpServletResponse;***

***import java.io.IOException;***

***import java.util.logging.FileHandler;***

***import java.util.logging.Level;***

***import java.util.logging.Logger;***

***import java.util.logging.SimpleFormatter;***

***public class Main {***

***public static void main(String[] args) throws Exception {***

***// Configurar el Logger***

***Logger logger = Logger.getLogger(Main.class.getName());***

***FileHandler fileHandler = new FileHandler("http.log");***

***fileHandler.setFormatter(new SimpleFormatter());***

***logger.addHandler(fileHandler);***

***// Crear el servidor Jetty***

***Server server = new Server(8080); // Puerto 8080***

***server.setHandler(new SimpleHandler(logger));***

***// Iniciar el servidor***

***server.start();***

***server.join();***

***}***

***// Manejador de solicitudes HTTP***

***static class SimpleHandler extends AbstractHandler {***

***private final Logger logger;***

***public SimpleHandler(Logger logger) {***

***this.logger = logger;***

***}***

***@Override***

***public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {***

***// Loggear la solicitud HTTP***

***logger.log(Level.INFO, String.format("Solicitud recibida: %s %s", request.getMethod(), request.getRequestURI()));***

***// Configurar la respuesta***

***response.setContentType("text/html;charset=utf-8");***

***response.setStatus(HttpServletResponse.SC\_OK);***

***baseRequest.setHandled(true);***

***response.getWriter().println("<h1>Hello, Jetty!</h1>");***

***}***

***}***

***}***

*Este código crea un servidor Jetty en el puerto 8080 y configura un manejador de logs que escribe en el archivo "http.log". Cada vez que se recibe una solicitud HTTP, se registra un mensaje de log con el método y la URI de la solicitud. Se puede personalizar el formato y el nivel de los logs según tus necesidades.*

*Con este enfoque, se puede tener un registro detallado de todas las solicitudes que recibe el servidor HTTP.*

**Ejercicio**

Qué produce el siguiente appender ? Codifique y de un ejemplo de salida

|  |
| --- |
| <Configuration status="WARN">  <Appenders>  <Console name="Console" target="SYSTEM\_OUT">  <HTMLLayout>  </HTMLLayout>  </Console>  </Appenders>  <Loggers>  <Root level="info">  <AppenderRef ref="Console"/>  </Root>  </Loggers>  </Configuration> |

*El appender definido en la configuración XML proporcionada enviará mensajes de logging al sistema de salida estándar (stdout) utilizando un diseño HTML para formatear los mensajes de registro.*

***<?xml version="1.0" encoding="UTF-8"?>***

***<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"***

***"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">***

***<html xmlns="http://www.w3.org/1999/xhtml">***

***<head>***

***<title>Logging Output</title>***

***<style type="text/css">***

***<!--***

***body, table {font-family: arial,sans-serif; font-size: small;}***

***th {background: #99CCFF; font-size: small; font-family: Arial, sans-serif;}***

***-->***

***</style>***

***</head>***

***<body>***

***<table cellspacing="0" cellpadding="2" border="1" bordercolor="#224466" width="100%">***

***<tr>***

***<th>Time</th>***

***<th>Thread</th>***

***<th>Level</th>***

***<th>Logger</th>***

***<th>Message</th>***

***</tr>***

***<tr>***

***<td>2024-04-19 12:30:45,123</td>***

***<td>main</td>***

***<td>INFO</td>***

***<td>com.example.MyClass</td>***

***<td>This is a sample log message.</td>***

***</tr>***

***</table>***

***</body>***

***</html>***

*En este ejemplo, un mensaje de logging con nivel INFO se ha formateado en una tabla HTML con columnas para la hora, el hilo, el nivel de logging, el logger y el mensaje. Esto es lo que se imprimiría en la consola o en el registro de un archivo, dependiendo de cómo esté configurado el sistema de registro.*

**Ejercicio**

Cree una aplicación que envíe mensajes de logging a un fichero, el cual es rotado cada 2 minutos.

Crear un archivo de configuración dónde utilicemos un “RollingFile” para escribir los logs y configurar la rotación del archivo cada 2 minutos.

*Primero, se necesita agregar Log4j a tus dependencias. Si estás utilizando Maven, puedes agregarlo al archivo pom.xml de la siguiente manera:*

***<dependency>***

***<groupId>org.apache.logging.log4j</groupId>***

***<artifactId>log4j-core</artifactId>***

***<version>2.17.1</version> <!-- La versión más reciente al momento de escribir esto -->***

***</dependency>***

*Luego, crea un archivo de configuración de Log4j (log4j2.xml) donde configuraremos un "RollingFile" para escribir los logs y establecer la rotación del archivo cada 2 minutos. Aquí está el contenido del archivo log4j2.xml:*

***<?xml version="1.0" encoding="UTF-8"?>***

***<Configuration status="WARN">***

***<Appenders>***

***<RollingFile name="File" fileName="logs/app.log"***

***filePattern="logs/app-%d{yyyy-MM-dd-HH-mm}.log">***

***<PatternLayout>***

***<pattern>%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n</pattern>***

***</PatternLayout>***

***<Policies>***

***<TimeBasedTriggeringPolicy interval="2" modulate="true"/>***

***</Policies>***

***</RollingFile>***

***</Appenders>***

***<Loggers>***

***<Root level="info">***

***<AppenderRef ref="File"/>***

***</Root>***

***</Loggers>***

***</Configuration>***

*Ahora, se puede crear una aplicación Java que utilice esta configuración para enviar mensajes de logging a un archivo que se rota cada 2 minutos. Aquí tienes un ejemplo de aplicación:*

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

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

***public class Main {***

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

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

***// Envía algunos mensajes de logging para probar***

***for (int i = 0; i < 10; i++) {***

***logger.info("Este es un mensaje de prueba número " + (i + 1));***

***try {***

***Thread.sleep(1000); // Espera 1 segundo***

***} catch (InterruptedException e) {***

***e.printStackTrace();***

***}***

***}***

***}***

***}***

*Este código creará un archivo de registro app.log en el directorio logs dentro del directorio de trabajo de la aplicación. Cada 2 minutos, este archivo se rotará y se creará uno nuevo con un nombre que incluye la fecha y hora actual. Los mensajes de log se escribirán en este archivo utilizando el formato especificado en el archivo de configuración log4j2.xml.*

*Hay que asegurarse de tener un directorio logs en tu proyecto para que los archivos de registro se puedan crear correctamente.*

**Ejercicio**

Modifique la aplicación anterior para que el fichero sea rotado según una expresión cron. (Coloque un horario válido para probar que funciona)

*Para rotar el archivo de registro según una expresión cron en Log4j 2, puedes usar la clase CronTriggeringPolicy. Aquí tienes cómo modificar la configuración y el código para hacerlo:*

*Modifica el archivo de configuración log4j2.xml para utilizar CronTriggeringPolicy en lugar de TimeBasedTriggeringPolicy*

***<?xml version="1.0" encoding="UTF-8"?>***

***<Configuration status="WARN">***

***<Appenders>***

***<RollingFile name="File" fileName="logs/app.log"***

***filePattern="logs/app-%d{yyyy-MM-dd-HH-mm}.log">***

***<PatternLayout>***

***<pattern>%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n</pattern>***

***</PatternLayout>***

***<Policies>***

***<CronTriggeringPolicy schedule="0 0/2 \* \* \* ?"/> //explica que se rota cada 2min***

***</Policies>***

***</RollingFile>***

***</Appenders>***

***<Loggers>***

***<Root level="info">***

***<AppenderRef ref="File"/>***

***</Root>***

***</Loggers>***

***</Configuration>***

*Actualizamos codigo:*

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

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

***public class Main {***

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

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

***// Envía algunos mensajes de logging para probar***

***for (int i = 0; i < 10; i++) {***

***logger.info("Este es un mensaje de prueba número " + (i + 1));***

***try {***

***Thread.sleep(1000); // Espera 1 segundo***

***} catch (InterruptedException e) {***

***e.printStackTrace();***

***}***

***}***

***}***

***}***

*Con esta configuración y código, el archivo de registro app.log debería rotarse cada 2 minutos según la expresión cron especificada en el archivo de configuración. Se puede cambiar la expresión cron según las necesidades específicas de rotación del archivo. Por ejemplo, para probar que funciona, podrías establecer la rotación del archivo cada minuto con la expresión cron 0 \* \* \* \* ?*

**Ejercicio**

Cree una aplicación para mostrar mensajes de logging al menos en cuatro formatos diferentes utilizando **PatternLayout**.

*Primero, asegúrate de tener Log4j 2 agregado a tus dependencias. Puedes agregarlo al archivo pom.xml si estás utilizando Maven:*

***<dependency>***

***<groupId>org.apache.logging.log4j</groupId>***

***<artifactId>log4j-core</artifactId>***

***<version>2.17.1</version> <!-- La versión más reciente al momento de escribir esto -->***

***</dependency>***

*Luego, crea un archivo de configuración de Log4j 2 (log4j2.xml) que especifique varios patrones de layout para diferentes tipos de mensajes de logging*

***<?xml version="1.0" encoding="UTF-8"?>***

***<Configuration status="WARN">***

***<Appenders>***

***<!-- Appender para formato simple -->***

***<Console name="SimpleConsole" target="SYSTEM\_OUT">***

***<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>***

***</Console>***

***<!-- Appender para formato detallado -->***

***<Console name="DetailedConsole" target="SYSTEM\_OUT">***

***<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>***

***</Console>***

***<!-- Appender para formato personalizado -->***

***<Console name="CustomConsole" target="SYSTEM\_OUT">***

***<PatternLayout pattern="%highlight{[%d{yyyy-MM-dd HH:mm:ss.SSS}][%t][%-5level][%logger{36}] - %msg%n}"/>***

***</Console>***

***<!-- Appender para formato JSON -->***

***<Console name="JsonConsole" target="SYSTEM\_OUT">***

***<JsonLayout complete="true" compact="true" eventEol="true"/>***

***</Console>***

***</Appenders>***

***<Loggers>***

***<Root level="info">***

***<!-- Usar los appenders definidos anteriormente -->***

***<AppenderRef ref="SimpleConsole"/>***

***<AppenderRef ref="DetailedConsole"/>***

***<AppenderRef ref="CustomConsole"/>***

***<AppenderRef ref="JsonConsole"/>***

***</Root>***

***</Loggers>***

***</Configuration>***

Luego en java creamos una aplicacion que utilice patrones de layout para enviar los mensajes de loggings:

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

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

***public class Main {***

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

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

***// Envía mensajes de logging utilizando diferentes formatos***

***logger.info("Mensaje de logging en formato simple.");***

***logger.error("¡Error! Mensaje de logging en formato detallado.");***

***logger.warn("¡Cuidado! Mensaje de logging en formato personalizado.");***

***logger.debug("Mensaje de logging en formato JSON.");***

***}***

***}***

*Con este código y la configuración log4j2.xml, cada mensaje de logging se mostrará en la consola en cuatro formatos diferentes:*

*Formato simple: [HH:mm:ss.SSS] [Hilo] NivelLogger - Mensaje*

*Formato detallado: [yyyy-MM-dd HH:mm:ss.SSS] [Hilo] NivelLogger - Mensaje*

*Formato personalizado: [yyyy-MM-dd HH:mm:ss.SSS][Hilo][NivelLogger][NombreClaseLogger] - Mensaje*

*Formato JSON: Formato JSON compacto con todas las propiedades del evento de logging.*

**Ejercicio**

Qué resultado produce el siguiente patrón:

1. %d [%-6p] %c{1} - %m%n
2. %sn %d{yyyy/MM/dd HH:mm:ss,SSS} %r [%-6p] [%t] %c{3} %C{3}.%M(%F:%L) - %m%n
3. **2024-04-25 14:30:00 [INFO ] MyApp - This is an informational message**
4. **1 2024/04/25 15:00:00,123 12345 [INFO ] [Thread-1] com.example.ClassName MethodName(SourceFile:123) - This is a log message**

**Ejercicio**

Investigue, documente y aplique un ejemplo de la clase ThreadContext.

**La clase ThreadContext es una parte de la biblioteca de Apache Log4j, que proporciona un contexto de hilo en el que se pueden almacenar y recuperar datos específicos del hilo en el contexto de registro (logging). Esto es útil cuando se desea asociar cierta información con un hilo específico, como un identificador de sesión, un nombre de usuario, o cualquier otro tipo de metadatos relevantes para el proceso de registro.**

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

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

**import org.apache.logging.log4j.ThreadContext;**

**public class ThreadContextExample {**

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

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

**// Establecer un valor en el contexto de hilo**

**ThreadContext.put("userId", "12345");**

**// Registrar un mensaje**

**logger.info("Este es un mensaje de registro");**

**// Eliminar el valor del contexto de hilo**

**ThreadContext.remove("userId");**

**// Registrar otro mensaje**

**logger.info("Este es otro mensaje de registro");**

**}**

**}**