**Log4j2**

**1. Introduction to Log4j2**

**Apache Log4j2** is a popular logging library for Java applications that is part of the Apache Logging Services. It provides a fast, flexible, and reliable way to log messages from applications. Log4j2 improves on the original Log4j and Logback by providing:

* **Asynchronous logging**: It allows logging messages asynchronously, which significantly improves performance, especially in high-traffic applications.
* **Customizable logging levels**: You can define different logging levels such as TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.
* **Multiple appenders**: Log4j2 can log messages to multiple destinations such as the console, files, databases, and remote servers.
* **Flexible configuration**: Log4j2 allows configuration via XML, JSON, YAML, and properties files.

**2. Log4j2 Features**

* **Log Levels**: Log4j2 supports six different log levels to help control the verbosity of the logs.
  + TRACE: Most verbose, detailed logs for tracing execution.
  + DEBUG: Less detailed than TRACE, used during development.
  + INFO: Used for general messages about the application's operation.
  + WARN: Logs warning messages indicating potential issues.
  + ERROR: Logs error messages, typically used when something goes wrong.
  + FATAL: Used for severe error messages where the application cannot continue.
* **Appendices**: Log4j2 allows you to log messages to multiple destinations, such as:
  + Console
  + File
  + Database
  + Socket, JMS, and more.
* **Asynchronous Logging**: Log4j2 supports asynchronous logging, improving performance by offloading logging to a separate thread.

**3. Setting Up Log4j2 in a Maven Project**

1. **Add Dependencies** to pom.xml for Maven projects:

<dependencies>

<!-- Log4j2 API -->

<dependency>

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

<artifactId>log4j-api</artifactId>

<version>2.17.1</version> <!-- Use the latest version -->

</dependency>

<!-- Log4j2 Core -->

<dependency>

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

<artifactId>log4j-core</artifactId>

<version>2.17.1</version> <!-- Use the latest version -->

</dependency>

</dependencies>

1. **Create log4j2.properties Configuration File**

This configuration file defines the logging setup for your application. Place this file under src/main/resources.

# Log4j2 Properties Configuration

# General configuration

status = warn

name = PropertiesConfig

# Console Appender (for logging to console)

appender.console.type = Console

appender.console.name = ConsoleAppender

appender.console.layout.type = PatternLayout

appender.console.layout.pattern = %d{ISO8601} [%t] %-5level %logger{36} - %msg%n

# File Appender (for logging to a file)

appender.file.type = File

appender.file.name = FileAppender

appender.file.fileName = logs/app.log

appender.file.layout.type = PatternLayout

appender.file.layout.pattern = %d{ISO8601} [%t] %-5level %logger{36} - %msg%n

# Root Logger Configuration

rootLogger.level = debug

rootLogger.appenderRef.console.ref = ConsoleAppender

rootLogger.appenderRef.file.ref = FileAppender

**4. Log4j2 Configuration Breakdown**

* **status**: Defines the internal logging level of Log4j2 itself. Typically set to warn or error unless you want to debug the logging framework itself.
* **appender.console**: Configures the console appender, which will print logs to the console with a PatternLayout (a custom format for the logs).
* **appender.file**: Configures the file appender, which writes log messages to a file (app.log inside the logs directory).
* **rootLogger**: Defines the root logger's logging level (debug) and links it to both the console and file appenders.
* Log4j2 supports the following levels, ordered by increasing severity:
* TRACE < DEBUG < INFO < WARN < ERROR < FATAL
* If the root logger is set to DEBUG, then logs at DEBUG, INFO, WARN, ERROR, and FATAL levels will be printed. If the level is set to INFO, then logs at INFO, WARN, ERROR, and FATAL will be printed, but TRACE and DEBUG messages will be ignored.

**5. Sample Java Program to Demonstrate Log4j2 Logging**

This Java program demonstrates logging at different levels and directs the logs to both the console and a log file (app.log).

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

public class Log4j2Demo {

// Creating a logger instance for the class

private static final Logger logger = LogManager.getLogger(Log4j2Demo.class);

public static void main(String[] args) {

// TRACE level - Most verbose, typically used for development and debugging.

logger.trace("This is a TRACE level message.");

// DEBUG level - Used for debugging purposes, not as verbose as TRACE.

logger.debug("This is a DEBUG level message.");

// INFO level - Used to log general information about the program's execution.

logger.info("This is an INFO level message.");

// WARN level - Used for potentially harmful situations.

logger.warn("This is a WARN level message.");

// ERROR level - Used to log error events that might allow the application to continue running.

logger.error("This is an ERROR level message.");

// FATAL level - Used to log very severe error events that lead to application termination.

logger.fatal("This is a FATAL level message.");

}

}

**6. Explanation of the Program**

* **Logger Instance**: The LogManager.getLogger(Log4j2Demo.class) creates a logger instance specific to the class. You can also create loggers for specific components/modules of the application.
* **Logging Levels**: The program logs messages using all six log levels: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.
* **Log Output**: The logs will be printed to both the console and saved to a file (logs/app.log) with a custom pattern.

**7. Example Output**

**Console Output:**

2025-02-15 10:15:32,123 [main] TRACE Log4j2Demo - This is a TRACE level message.

2025-02-15 10:15:32,124 [main] DEBUG Log4j2Demo - This is a DEBUG level message.

2025-02-15 10:15:32,124 [main] INFO Log4j2Demo - This is an INFO level message.

2025-02-15 10:15:32,125 [main] WARN Log4j2Demo - This is a WARN level message.

2025-02-15 10:15:32,126 [main] ERROR Log4j2Demo - This is an ERROR level message.

2025-02-15 10:15:32,126 [main] FATAL Log4j2Demo - This is a FATAL level message.

**Log File (logs/app.log):**

2025-02-15 10:15:32,123 [main] TRACE Log4j2Demo - This is a TRACE level message.

2025-02-15 10:15:32,124 [main] DEBUG Log4j2Demo - This is a DEBUG level message.

2025-02-15 10:15:32,124 [main] INFO Log4j2Demo - This is an INFO level message.

2025-02-15 10:15:32,125 [main] WARN Log4j2Demo - This is a WARN level message.

2025-02-15 10:15:32,126 [main] ERROR Log4j2Demo - This is an ERROR level message.

2025-02-15 10:15:32,126 [main] FATAL Log4j2Demo - This is a FATAL level message.

**8. Other Logging Features**

* **Asynchronous Logging**: Log4j2 supports asynchronous logging, allowing logs to be written in a separate thread. This can improve performance in applications with heavy logging needs. Enable asynchronous logging by using the AsyncLogger and AsyncAppender.

Example:

appender.async.type = Async

appender.async.appenderRef.console.ref = ConsoleAppender

* **Rolling File Appender**: Log4j2 allows you to configure rolling file appenders to rotate logs based on size or time.

Example for rolling based on size:

appender.rolling.type = RollingFile

appender.rolling.name = RollingFileAppender

appender.rolling.fileName = logs/app.log

appender.rolling.filePattern = logs/app-%d{MM-dd-yyyy}.log

appender.rolling.policy.type = SizeBasedTriggeringPolicy

appender.rolling.policy.size = 10MB