**Logging -Log4J**

**📌 Why is Logging Important?**

Logging helps in:

✅ Debugging and troubleshooting applications

✅ Monitoring application behaviour

✅ Auditing and tracking system events

**📌 Installing Log4j**

**Option 1: Maven Users (Recommended)**

Add this dependency to pom.xml:

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

**Option 2: Manual Download**

1. Download **log4j-1.2.17.jar** from [Apache Log4j Website](https://logging.apache.org/log4j/1.2/)
2. Add it to your project's lib folder.
3. Add it to your **classpath**.

**Basic Logging in Log4j**

**📌 Log4j Components**

✔ **Logger** – Generates log messages

✔ **Appender** – Defines where logs are sent (Console, File, Database)

✔ **Layout** – Specifies log format

**📌 Logging Levels in Log4j**

| **Level** | **Purpose** |
| --- | --- |
| FATAL | Critical errors that cause shutdown |
| ERROR | Significant errors in execution |
| WARN | Potential issues (not errors yet) |
| INFO | General application events |
| DEBUG | Detailed debugging messages |
| TRACE | Most fine-grained debugging messages |

**📌 Writing Basic Logs**

org.apache.log4j.Logger;

public class Log4jBasicExample {

private static final Logger logger = Logger.getLogger(Log4jBasicExample.class);

public static void main(String[] args) {

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

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

logger.warn("This is a WARNING message");

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

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

}

}

**Sample log4j.properties File**

properties

CopyEdit

# 1️⃣ Root Logger Configuration

log4j.rootLogger=DEBUG, console, file

# 2️⃣ Console Appender

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d [%p] %m%n

# 3️⃣ File Appender

log4j.appender.file=org.apache.log4j.FileAppender

log4j.appender.file.File=logs/app.log

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d [%p] %m%n

**📖 Line-by-Line Explanation**

**1️⃣ Root Logger Configuration**

log4j.rootLogger=DEBUG, console, file

**🔍 Explanation**

* log4j.rootLogger → Defines the **root logger** (global logger).
* DEBUG → The minimum **log level** to capture (DEBUG, INFO, WARN, ERROR, FATAL).
* console, file → Specifies **appenders** (log destinations).

👉 **This means:** All logs with level DEBUG or higher will be sent to the **console** and a **file**.

**2️⃣ Console Appender (Logging to Console)**

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

**Appender Configuration**

* log4j.appender.console=org.apache.log4j.ConsoleAppender
  + Creates a **console appender** to print logs in the terminal.

**Log Formatting**

* log4j.appender.console.layout=org.apache.log4j.PatternLayout
  + Specifies **how logs should be formatted**.
* log4j.appender.console.layout.ConversionPattern=%d [%p] %m%n
  + Defines the **log message format**:
    - %d → **Date & Time**
    - %p → **Log Level** (DEBUG, INFO, etc.)
    - %m → **Actual Log Message**
    - %n → **New Line**

🔹 **Example Output in Console:**

2025-03-01 12:00:00 [INFO] Application started

**3️⃣ File Appender (Logging to a File)**

log4j.appender.file=org.apache.log4j.FileAppender

log4j.appender.file.File=logs/app.log

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

**Appender Configuration**

* log4j.appender.file=org.apache.log4j.FileAppender
  + Creates a **file appender** to log messages into a file.
* log4j.appender.file.File=logs/app.log
  + Specifies the **log file location** (logs/app.log).

**Log Formatting (Same as Console)**

* log4j.appender.file.layout=org.apache.log4j.PatternLayout
  + Defines the log message format.
* log4j.appender.file.layout.ConversionPattern=%d [%p] %m%n
  + The log format remains the same.

🔹 **Example Output in logs/app.log:**

25-03-01 12:00:00 [INFO] Application started

**🛠️ Additional Configurations**

**📌 Log Rotation (RollingFileAppender)**

If you don’t want your log file to grow infinitely, use RollingFileAppender:

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=logs/app.log

log4j.appender.file.MaxFileSize=1MB

log4j.appender.file.MaxBackupIndex=5

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* MaxFileSize=1MB → When the log file reaches 1MB, it creates a **new** log file.
* MaxBackupIndex=5 → Keeps up to **5 old log files** (app.log.1, app.log.2, etc.).

**📌 Logging to a Database (JDBCAppender)**

If you want to store logs in a database:

log4j.appender.db=org.apache.log4j.jdbc.JDBCAppender

log4j.appender.db.URL=jdbc:mysql://localhost:3306/logsdb

log4j.appender.db.driver=com.mysql.jdbc.Driver

log4j.appender.db.user=root

log4j.appender.db.password=root

log4j.appender.db.sql=INSERT INTO logs (log\_date, level, message) VALUES ('%d', '%p', '%m')

log4j.appender.db.layout=org.apache.log4j.PatternLayout

**🔍 Explanation**

* Logs messages into a **MySQL database** instead of a file.

**📌 Logging Different Parts of an Application**

If you want **different log levels** for different packages:

log4j.logger.com.myapp.controllers=INFO, console

log4j.logger.com.myapp.services=DEBUG, file

**🔍 Explanation**

* Logs in com.myapp.controllers will be at **INFO level**.
* Logs in com.myapp.services will be at **DEBUG level**.

**Log4j Appender Types Explained**

In **Log4j**, an **Appender** is responsible for **sending log messages** to a particular destination (console, file, database, etc.). There are **multiple types** of appenders in Log4j, each serving a different purpose.

**📌 1. ConsoleAppender (Logs to Console)**

Used to **print logs on the console (stdout/stderr)**. Helpful for **development and debugging**.

**Configuration**

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* Logs messages to the **console (System.out)**.
* Uses **PatternLayout** to format log messages.

**📝 Example Output in Console**

25-03-01 12:00:00 [INFO] Application started

**📌 2. FileAppender (Logs to a File)**

Used to **store logs in a file** for analysis.

**Configuration**

log4j.appender.file=org.apache.log4j.FileAppender

log4j.appender.file.File=logs/app.log

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* Logs messages to a **file (logs/app.log)**.
* **PatternLayout** formats the messages.

**📝 Example Output in app.log**

2025-03-01 12:00:00 [INFO] Application started

**📌 3. RollingFileAppender (Logs to a Rotating File)**

Used to **rotate log files** when they reach a certain size, preventing **log overflow**.

**Configuration**

log4j.appender.rollingFile=org.apache.log4j.RollingFileAppender

log4j.appender.rollingFile.File=logs/app.log

log4j.appender.rollingFile.MaxFileSize=1MB

log4j.appender.rollingFile.MaxBackupIndex=5

log4j.appender.rollingFile.layout=org.apache.log4j.PatternLayout

log4j.appender.rollingFile.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* MaxFileSize=1MB → Log file rotates after reaching **1MB**.
* MaxBackupIndex=5 → Keeps **5 backup logs** (app.log.1, app.log.2, etc.).

**📌 4. DailyRollingFileAppender (Logs to a File Daily)**

Used to create **a new log file every day**.

**Configuration**

log4j.appender.dailyFile=org.apache.log4j.DailyRollingFileAppender

log4j.appender.dailyFile.File=logs/app.log

log4j.appender.dailyFile.DatePattern='.'yyyy-MM-dd

log4j.appender.dailyFile.layout=org.apache.log4j.PatternLayout

log4j.appender.dailyFile.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* DatePattern='.'yyyy-MM-dd → Creates a **new log file daily** (app.log.2025-03-01).

**📌 5. JDBCAppender (Logs to a Database)**

Used to **store logs in a database** instead of a file.

**Configuration**

log4j.appender.database=org.apache.log4j.jdbc.JDBCAppender

log4j.appender.database.URL=jdbc:mysql://localhost:3306/logsdb

log4j.appender.database.driver=com.mysql.jdbc.Driver

log4j.appender.database.user=root

log4j.appender.database.password=root

log4j.appender.database.sql=INSERT INTO logs (log\_date, level, message) VALUES ('%d', '%p', '%m')

log4j.appender.database.layout=org.apache.log4j.PatternLayout

**🔍 Explanation**

* Stores logs in a **MySQL database**.
* sql=INSERT INTO logs... → Defines how logs are stored.

**📌 6. SMTPAppender (Sends Logs via Email)**

Used to **send logs via email** when critical errors occur.

**Configuration**

log4j.appender.email=org.apache.log4j.net.SMTPAppender

log4j.appender.email.SMTPHost=smtp.example.com

log4j.appender.email.From=alerts@example.com

log4j.appender.email.To=admin@example.com

log4j.appender.email.Subject=Critical Error in Application

log4j.appender.email.BufferSize=10

log4j.appender.email.layout=org.apache.log4j.PatternLayout

log4j.appender.email.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* **Sends an email** when ERROR or FATAL logs occur.
* BufferSize=10 → Sends **email after 10 log messages**.

**📌 7. SyslogAppender (Logs to Syslog - UNIX/Linux)**

Used to send logs to **syslog servers** in Unix/Linux environments.

**Configuration**

log4j.appender.syslog=org.apache.log4j.net.SyslogAppender

log4j.appender.syslog.SyslogHost=localhost

log4j.appender.syslog.Facility=LOCAL1

log4j.appender.syslog.layout=org.apache.log4j.PatternLayout

log4j.appender.syslog.layout.ConversionPattern=%d [%p] %m%n

**🔍 Explanation**

* SyslogHost=localhost → Sends logs to **local syslog**.
* Facility=LOCAL1 → Defines the syslog **facility level**.

**Log4j Methods in Java**

Log4j provides several methods to **log messages** at different levels, **configure logging**, and **manage loggers** dynamically. Below is a breakdown of **important Log4j methods** with explanations and examples.

**📌 1. Getting a Logger**

Before using Log4j, you need to **get an instance of a Logger**.

**Method**

static Logger getLogger(String name)

static Logger getLogger(Class clazz)

**Example**

import org.apache.log4j.Logger;

public class MyApp {

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

public static void main(String[] args) {

logger.info("Application started");

}

}

🔹 getLogger(MyApp.class) → Gets a **Logger** instance for the MyApp class.

**📌 2. Logging Methods (INFO, DEBUG, ERROR, etc.)**

These methods are used to log messages at different **log levels**.

| **Method** | **Description** |
| --- | --- |
| logger.debug("message") | Logs **detailed debug** messages. |
| logger.info("message") | Logs **informational messages**. |
| logger.warn("message") | Logs **warnings** (potential issues). |
| logger.error("message") | Logs **errors** (failures in code execution). |
| logger.fatal("message") | Logs **fatal errors** (serious application failures). |

**Example**

import org.apache.log4j.Logger;

public class LogExample {

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

public static void main(String[] args) {

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

logger.info("Application is running");

logger.warn("This is a warning!");

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

logger.fatal("Fatal error occurred!");

}

}

🔹 Each method logs messages at a **specific severity level**.

**📌 3. Logging with Exception Stack Trace**

You can **log exceptions** with detailed stack traces.

**Example**

try {

int result = 10 / 0; // Division by zero

} catch (Exception e) {

logger.error("An exception occurred: ", e);

}

🔹 This prints the **exception stack trace** to logs.

**📌 4. Checking if Logging is Enabled**

Before logging, you can check if a **specific log level is enabled**.

| **Method** | **Description** |
| --- | --- |
| logger.isDebugEnabled() | Returns true if **DEBUG** logging is enabled. |
| logger.isInfoEnabled() | Returns true if **INFO** logging is enabled. |
| logger.isWarnEnabled() | Returns true if **WARN** logging is enabled. |

**Example**

if (logger.isDebugEnabled()) {

logger.debug("Expensive debug operation");

}

🔹 This prevents **unnecessary log calculations** when DEBUG is disabled.

**📌 5. Changing Log Level Dynamically**

You can **change log levels at runtime**.

**Method**

import org.apache.log4j.Level;

import org.apache.log4j.Logger;

Logger logger = Logger.getLogger(MyApp.class);

logger.setLevel(Level.WARN); // Changes logging level to WARN

🔹 Now, only **WARN, ERROR, and FATAL** logs will be recorded.

**📌 6. Formatting Log Messages with Parameters**

Instead of **string concatenation**, use {} placeholders.

**Example**

username = "John";

logger.info("User {} logged in", username); // Incorrect in Log4j 1.x

// Correct way (Log4j 1.x doesn't support placeholders)

logger.info("User " + username + " logged in");

🔹 **Note**: Log4j 2 and SLF4J support {} placeholders.

**📌 7. Getting Parent Logger (Logger Hierarchy)**

Log4j loggers follow a **hierarchy**.

**Method**

Logger parentLogger = Logger.getLogger("com.myapp");

Logger childLogger = Logger.getLogger("com.myapp.module");

🔹 childLogger **inherits settings** from parentLogger.

**📌 8. Adding & Removing Appenders**

You can add or remove appenders **dynamically**.

**Methods**

| **Method** | **Description** |
| --- | --- |
| logger.addAppender(Appender appender) | Adds an appender (e.g., console, file). |
| logger.removeAppender(Appender appender) | Removes a specific appender. |
| logger.removeAllAppenders() | Removes all appenders. |

**Example**

import org.apache.log4j.ConsoleAppender;

import org.apache.log4j.SimpleLayout;

ConsoleAppender consoleAppender = new ConsoleAppender(new SimpleLayout());

logger.addAppender(consoleAppender);

🔹 Adds a **console appender** programmatically.

**📌 9. Configuring Logging Programmatically (Without log4j.properties)**

Instead of using log4j.properties, you can configure Log4j **in code**.

**Example**

import org.apache.log4j.\*;

public class LogConfigExample {

public static void main(String[] args) {

Logger logger = Logger.getLogger(LogConfigExample.class);

// Create Console Appender

ConsoleAppender consoleAppender = new ConsoleAppender(new PatternLayout("%d [%p] %m%n"));

logger.addAppender(consoleAppender);

// Set Log Level

logger.setLevel(Level.DEBUG);

logger.info("Logging configured programmatically!");

}

}

🔹 This **configures Log4j without log4j.properties**.

| **Feature** | **Log4j (1.x)** | **Log4j2** | **SLF4J** |
| --- | --- | --- | --- |
| Performance | Slow | Faster | Faster (with Logback) |
| Async Logging | ❌ No | ✅ Yes | ✅ Yes |
| Config File | log4j.properties | log4j2.xml | Uses Logback or Log4j2 |
| Thread Safety | ❌ No | ✅ Yes | ✅ Yes |

🔹 **SLF4J** is an abstraction for logging frameworks, allowing switching between Log4j a