Eg: Log4jHtmlLayoutFileAppender

Case-1:

# 1. Specify Logger level and appender

log4j.rootLogger=DEBUG, R

# 2. Define the appender

log4j.appender.R=org.apache.log4j.FileAppender

# 3. Specify the file where log will be written

log4j.appender.R.File=C:\\Users\\Administrator\\Downloads\\new1\\log.html

# 4. Enable appending to the file (true = keep old logs)

log4j.appender.R.append=true

# 5. Set the layout to HTMLLayout

log4j.appender.R.layout=org.apache.log4j.HTMLLayout

Case-2:

Eg: Log4jMultAppndConsoleAppndSimpleLayoutAndFileAppndHtmlLayout

# 1. Specify Logger level and multiple appenders (Console + File)

log4j.rootLogger=DEBUG, R, C

# 2. Configure ConsoleAppender with SimpleLayout

log4j.appender.C=org.apache.log4j.ConsoleAppender

log4j.appender.C.layout=org.apache.log4j.SimpleLayout

# 3. Configure FileAppender with HTMLLayout

log4j.appender.R=org.apache.log4j.FileAppender

log4j.appender.R.File=C:\\Users\\Administrator\\Downloads\\new1\\log1.html

log4j.appender.R.append=true

log4j.appender.R.layout=org.apache.log4j.HTMLLayout

Case3: working with file appender and xmllayout

Eg: Log4jFileappenderXmlLayout

# Specify Logger level and appender

log4j.rootLogger=INFO, R

# Specify FileAppender

log4j.appender.R=org.apache.log4j.FileAppender

# Specify the file to write logs

log4j.appender.R.File=C:\\Users\\Administrator\\Downloads\\new1\\info.xml

# Enable append mode

log4j.appender.R.append=true

# Specify layout as XMLLayout

log4j.appender.R.layout=org.apache.log4j.xml.XMLLayout

Case 4: For RollingFileAppender and PatternLayout

Eg: Log4jRollingFileAppenderPatternLayout

log4j.rootLogger=DEBUG, R

# Define the RollingFileAppender

log4j.appender.R=org.apache.log4j.RollingFileAppender

log4j.appender.R.File=log\_info.txt

log4j.appender.R.MaxFileSize=5KB

log4j.appender.R.MaxBackupIndex=3

log4j.appender.R.append=true

# Use PatternLayout with a detailed conversion pattern

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

log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1} %M [%t] %L - %m%n

case 5:

eg: DailyRollingFileAppenderPatternLayout

log4j.rootLogger=DEBUG, R

# Define the DailyRollingFileAppender

log4j.appender.R=org.apache.log4j.DailyRollingFileAppender

log4j.appender.R.File=log\_info1.txt

log4j.appender.R.append=true

log4j.appender.R.datePattern='.'yyyy-MM-dd-HH-mm

# Define the layout

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

log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1} %M [%t] %L - %m%n

**1. Default Logging in Spring Boot**

* When you create a Spring Boot project, you **do not need to add any logging library** — Spring Boot already has logging built in.
* **By default**, Spring Boot uses:
  + **SLF4J** → This stands for *Simple Logging Facade for Java*.
  + **Logback** → This is the actual logging engine that writes the log messages.

**What is SLF4J?**

* Think of SLF4J as a **middleman** or **universal remote control** for logging.
* It does **not** create or store log messages by itself.
* Instead, it just provides a **common way** for your code to send log messages.
* SLF4J then passes these log messages to whatever logging tool is connected to it (like **Logback**, **Log4j**, or others).

**Why use SLF4J?**

* If your code uses SLF4J, you can **easily change the logging framework** later (from Logback to Log4j, for example) without rewriting your code.
* Your code always talks to SLF4J, and SLF4J talks to the actual logging tool.

**Example flow (Default in Spring Boot):**

1. Your code → logger.info("Hello World")
2. SLF4J gets the message → "Hello World"
3. SLF4J passes it to Logback (default)
4. Logback decides how to display or save it (console, file, etc.)

### ****2. If You Want to Use Log4j Instead****

Spring Boot does **not** use Log4j by default.  
If you want to use **Log4j 2**, you have to remove the default logging dependency and add Log4j dependencies.

#### **Step 1 – Remove Default Logging**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

#### **Step 2 – Add Log4j2 Dependency**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

### ****3. Log4j2 Configuration File****

Create a file **log4j2.xml** (or .properties) inside src/main/resources:

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

<Configuration status="WARN">

<Appenders>

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

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

</Console>

</Appenders>

<Loggers>

<Root level="info">

<AppenderRef ref="Console"/>

</Root>

</Loggers>

</Configuration>

### ****4. Using the Logger in Code****

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@Controller

public class TestController {

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

@GetMapping("/test")

public String testLogging() {

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

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

return "Check logs in console!";

}

}

### ****Summary****

* **Default**: Spring Boot → SLF4J + Logback
* **To use Log4j**: Remove default logging → Add spring-boot-starter-log4j2 → Create log4j2.xml → Use LoggerFactory in code.

**📝 Log4j 2 RollingFileAppender + PatternLayout (Spring Boot MVC)**

**1. Purpose**

* **RollingFileAppender** → Logs to a file and automatically rolls over based on size or time.
* **PatternLayout** → Defines how the log message looks (format).
* Used in **non-REST** projects with @Controller and JSPs.

**2. Key Files**

| **File** | **Purpose** |
| --- | --- |
| pom.xml | Add dependencies (Spring Boot Web, JSP, Log4j 2). |
| log4j2.xml | Log configuration (appenders, patterns, rolling rules). |
| application.properties | Set JSP prefix/suffix for view resolution. |
| Controller Class | Write logs and return view names. |
| JSP | Show UI, logs appear in console and file. |

**3. File Paths in log4j2.xml**

* **Relative path** → fileName="logs/app.log" → stored inside project folder when running.
* **Absolute path** → fileName="C:/mylogs/app.log" (Windows) or /var/log/myapp/app.log (Linux).
* Pattern for rolled files:

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

→ Creates timestamped logs, compressed as .gz.

**4. PatternLayout Format**

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

* %d{} → Date/time format.
* %t → Thread name.
* %level → Log level (INFO, ERROR, etc.).
* %logger{36} → Logger name, max length 36.
* %msg → The actual log message.
* %n → New line.

**5. Rolling Policies**

In <Policies> block:

* **SizeBasedTriggeringPolicy** → Rolls log after reaching a set size.

<SizeBasedTriggeringPolicy size="5MB"/>

* **TimeBasedTriggeringPolicy** → Rolls after a set time interval.

<TimeBasedTriggeringPolicy interval="1" modulate="true"/>

* + interval="1" → Roll every 1 time unit (minutes/hours/days depending on %d{} in filePattern).
  + modulate="true" → Aligns to time boundaries.

**6. Typical Controller Use**

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

@GetMapping("/test")

public String testLog() {

logger.info("Info message");

logger.error("Error message");

return "test"; // Goes to /WEB-INF/views/test.jsp

}

**7. JSP Location**

Set in application.properties:

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

Place JSP in:

src/main/webapp/WEB-INF/views/test.jsp

**8. Common Issues**

| **Problem** | **Cause** | **Fix** |
| --- | --- | --- |
| Log not writing to file | Wrong file path or missing folder | Create folder or give absolute path |
| Console only, no file | RollingFileAppender not in <Root> | Add <AppenderRef ref="RollingFileAppender"/> |
| JSP not loading | Prefix/suffix misconfigured | Fix in application.properties |
| Logs overwrite instead of roll | Missing <Policies> | Add size/time policies |

**9. Quick Example: RollingFileAppender**

<RollingFile name="RollingFileAppender"

fileName="logs/app.log"

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

<PatternLayout>

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

</PatternLayout>

<Policies>

<SizeBasedTriggeringPolicy size="5MB"/>

<TimeBasedTriggeringPolicy interval="1" modulate="true"/>

</Policies>

</RollingFile>

If you keep this note, you’ll never have to search again for:

* Where to put the file path
* How to set rolling rules
* How to connect logs + JSP in MVC

**📒 Notes on Console Appender & PatternLayout (Log4j2)**

**1️⃣ Console Appender**

* **Purpose**: Sends log messages to the console/terminal instead of a file.
* **When to use**: During development/debugging to see logs instantly without opening log files.
* **Key Property**:

<Console name="ConsoleAppender" target="SYSTEM\_OUT">

...

</Console>

* + target="SYSTEM\_OUT" → prints to standard output (normal logs).
  + target="SYSTEM\_ERR" → prints to error output (useful for only errors).

**2️⃣ PatternLayout**

* **Purpose**: Decides **how** the log message looks (format).
* **Example**:

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

* **Pattern parts**:
  + %d{...} → Date & time format.
  + %-5level → Log level (INFO, ERROR, etc.), padded for alignment.
  + %logger{36} → Logger name (class name, limited to 36 chars).
  + %msg → The log message.
  + %n → New line.

**3️⃣ Why Console Appender + PatternLayout is useful**

* Easy to **read logs** in a structured way.
* Helps quickly **debug during development**.
* Prevents unnecessary file writes when you just need temporary testing.

**📒 Notes on @Controller and @ResponseBody**

**1️⃣ Why @ResponseBody is needed with @Controller**

* @Controller in Spring MVC is meant for **returning view names** (like JSP, Thymeleaf, etc.).
* If you return a string in a normal @Controller method **without** @ResponseBody, Spring will think:

"Oh, this string is the name of a view (like test.jsp), so I should go find a JSP file with that name."

* Since "Check logs in console" is **not** a JSP file name, Spring will throw an error saying **view not found**.

**2️⃣ When you don’t need @ResponseBody**

* If you were **actually returning a JSP page** (like return "myPage";) and had a myPage.jsp in your WEB-INF/views/ folder, you would **not** use @ResponseBody.
* In that case, Spring resolves "myPage" → /WEB-INF/views/myPage.jsp.

**💡 Simple Rule**

* **Returning JSP page** → use @Controller (no @ResponseBody).
* **Returning plain text or JSON** → use @RestController **or** @Controller + @ResponseBody.

Eg: SpringBootConsoleAppenderPatternLayout

**📒 Daily Rolling File Appender – Notes**

* **Purpose:**  
  Automatically creates a **new log file at a set time interval** (e.g., daily at midnight) without overwriting the old one.
* **Log4j2 Implementation:**  
  Done using the <RollingFile> appender with a **TimeBasedTriggeringPolicy** in log4j2.xml.
* **When to Use:**
  + Long-running applications that need daily logs for auditing.
  + Projects where log history is important for debugging or compliance.
* **Advantages:**
  + Keeps logs organized by date.
  + Prevents one giant log file from growing endlessly.
  + Makes it easier to archive, compress, or delete old logs.
* **Key Config Parameters:**
  + fileName → current log file.
  + filePattern → naming format for rolled files (e.g., app-%d{yyyy-MM-dd}.log).
  + <TimeBasedTriggeringPolicy> → defines the time interval for rolling (1 = daily).
  + <PatternLayout> → controls log message format.
* **Common Pitfall:**  
  Make sure the log folder exists or Log4j2 may fail to write logs.

**🎨 PatternLayout – Notes**

* **Purpose:**  
  Controls the **exact text format** of each log line.
* **Why Important:**  
  Well-formatted logs are easier to read, search, and parse by log analysis tools.
* **Common Conversion Patterns:**
  + %d{yyyy-MM-dd HH:mm:ss} → date & time.
  + %p → log level (INFO, DEBUG, ERROR).
  + %c → logger name or class.
  + %m → actual log message.
  + %n → new line.
* **Example Pattern:**

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

* + Shows date, thread, log level, logger name, and message.
* **Tip:**  
  Avoid overly long logger names by using %logger{36}.

**⚡ Typical Daily Rolling File + Pattern Layout Setup (Log4j2)**

<Configuration status="WARN">

<Appenders>

<RollingFile name="DailyRollingFile"

fileName="logs/app.log"

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

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

<Policies>

<TimeBasedTriggeringPolicy interval="1"/>

</Policies>

</RollingFile>

</Appenders>

<Loggers>

<Root level="info">

<AppenderRef ref="DailyRollingFile"/>

</Root>

</Loggers>

</Configuration>

**📝 Quick Summary Table**

| **Feature** | **Daily Rolling File Appender** | **PatternLayout** |
| --- | --- | --- |
| **Purpose** | Roll logs daily | Format log messages |
| **Config Key** | <RollingFile> | <PatternLayout> |
| **Key Benefit** | Organized log history | Readable, structured logs |
| **Important Settings** | filePattern, <TimeBasedTriggeringPolicy> | %d, %p, %c, %m, %n |
| **When to Use** | Daily auditing, long-running apps | Any log output |

Eg: SpringBootDailyRollingFileAppenderPatternLayout