**Logger Usage Best Practices in Java Spring Boot Applications**

### 1. **Overview**

Logging is an essential part of any Spring Boot application. It provides insight into the behavior of the application, helps in debugging, monitoring, and troubleshooting issues.

Spring Boot uses SLF4J as the logging facade. While the default implementation is Logback, you can also use **Apache Log4j2** for advanced features and flexibility.

### 1.1 **Problems in Current Logging Mechanism**

1. **Proper Logs were not present** – Insufficient logging made debugging production issues difficult.
2. **Incorrect Logging Levels** – Logs were often marked at incorrect severity (e.g., INFO used instead of ERROR).
3. **No Runtime Control of Log Level** – Unable to dynamically modify log levels without restarting the service.
4. **Logs Restricted to Production Server** – Logs were not being pushed outside the production server, limiting access for support and monitoring teams.
5. **Lack of Monitoring Framework** – No centralized system for visualizing, filtering, or analyzing logs across environments.

### 2. **Logger Setup with Apache Log4j2**

#### A. Maven Dependencies

Use the latest Log4j2 core and API libraries (version 2.20.0). Also, exclude the default Logback:

<dependencies>  
 <!-- Exclude 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>  
  
 <!-- Apache Log4j2 Core and API -->  
 <dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-core</artifactId>  
 <version>2.20.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-api</artifactId>  
 <version>2.20.0</version>  
 </dependency>  
  
 <!-- Spring Boot bridge for Log4j2 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-log4j2</artifactId>  
 </dependency>  
</dependencies>

#### B. Logger Declaration

import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
@Service  
public class MyService {  
 private static final Logger logger = LogManager.getLogger(MyService.class);  
  
 public void process() {  
 logger.info("Processing started");  
 }  
}

### 3. **Logging Levels and Usage**

| Level | Description | When to Use |
| --- | --- | --- |
| TRACE | Very fine-grained info | For deep debugging in development only |
| DEBUG | Debug-level messages | To trace execution flow and variable values |
| INFO | General information | For normal application lifecycle events |
| WARN | Something unexpected but not an error | For deprecated APIs, unexpected states |
| ERROR | Error events that disrupt functionality | For exception handling and failures |
| FATAL | Severe errors causing system shutdown | For unrecoverable critical failures |

### 4. **Logging Configuration (log4j2.xml)**

Create a log4j2.xml file in 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>  
 <RollingFile name="FileAppender" fileName="logs/app.log" filePattern="logs/app-%d{yyyy-MM-dd}.log.gz">  
 <PatternLayout>  
 <Pattern>%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n</Pattern>  
 </PatternLayout>  
 <Policies>  
 <TimeBasedTriggeringPolicy interval="1" modulate="true"/>  
 <SizeBasedTriggeringPolicy size="10MB"/>  
 </Policies>  
 </RollingFile>  
 </Appenders>  
  
 <Loggers>  
 <Root level="info">  
 <AppenderRef ref="Console"/>  
 <AppenderRef ref="FileAppender"/>  
 </Root>  
 </Loggers>  
</Configuration>

### 5. **Best Practices**

* **Avoid String concatenation** in log statements:
* logger.debug("User id: {}", userId); // Good  
  logger.debug("User id: " + userId); // Bad
* **Use conditional logging** for expensive operations:
* if (logger.isDebugEnabled()) {  
   logger.debug("Result: {}", computeExpensiveResult());  
  }
* **Never log sensitive data** (passwords, tokens).
* **Structure logs consistently**: Include request ID, user ID, timestamp if applicable.
* **Use correlation IDs** across services to trace transactions.

### 6. **Advanced Features (Optional)**

* **Asynchronous logging** for high-performance applications.
* **Separate log files** by logger name or level.
* **JSON logging** for integration with log management tools.
* **Centralized Logging** using ELK, Graylog, or Splunk.

### 7. **Conclusion**

Using Apache Log4j2 with Spring Boot gives you fine-grained control over logging. Implement structured, consistent, and secure logging to support better debugging, monitoring, and production diagnostics.