Logování

Vytváření logů z aplikací slouží k zaznamenávání informací o průběhu programu, informací sloužících k ladění aplikace a také informací o výskytu chyb v aplikaci. Proč používat zvláštní knihovny pro logování, když můžeme stejných výsledků dosáhnout přímým zápisem zaznamů o činnosti aplikace do souboru nebo do standardního výstupu pomocí volání metody System.out.println(). Hlavním důvodem pro používání knihoven pro logování je jednoduchá parametrizovatelnost a jako další výhodu lze uvést snadnou dostupnost instance Loggeru (viz dále) ze všech tříd aplikace. Další ne nepodstatnou výhodou je i vysoká rychlost logování. Co je myšleno tou parametrizovatelností? Pokud ladíte pomocí výpisů metody System.out.println(), musíte před dokončením aplikace odstranit všechny nepotřebné výpisy. Při použití specializovaných prostředků pro logování stačí logování pomocí parametrického souboru vypnout. Stejně tak vám logování ušetří práci s nastavováním logovacích souborů atd.

Pro logování z aplikací napsaných v Javě máme několik možností. My si popíšeme dvě z nich. První je knihovna log4j, která je volně k dispozici na jakarta.apache.org. Druhou možností je použít prostředků pro logování implementovaných přímo ve standardu Javy a to od verze 1.4. Aplikace log4j je popsána podrobněji. Autoři s ní mají více zkušeností a i podle dostupných zdrojů je výkonnější něž logování z API Javy.

Log4J

Tuto aplikaci si můžete stáhnout ze serveru jakarta.apache.org, v současné době ve verzi 1.2.9. Na těchto stránkách je také k dispozici dokumentace k jednotlivým třídám, jednoduchý manuál a je zde uvedeno i několik odkazů na další zdroje informací. Jedinou nevýhodou log4j je, že podrobný manuál není volně k dispozici.

Základní pojmy používané v log4j

Koncepce log4j je založena na 6 základních pojmech:

priorita význam (důležitost) zprávy, každý Logger i Appender může mít nastaven práh (Threshold), od kterého dále zpracovává zprávy, pro označení priority se též používá pojem Level - úroveň (třída Level je potomkem třídy Priority, preferuje se použití třídy Level),

kategorie logická skupina, do které patří příslušná zpráva,

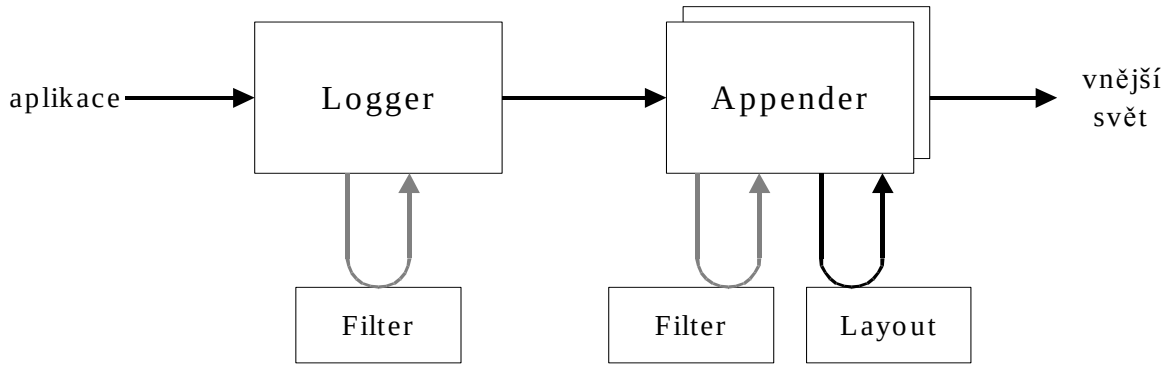
Logger objekt, pomocí kterého programátor vypisuje zprávy (volá příslušné metody tohoto objektu),

Appender objekt, který se stará o vlastní výpis, může existovat více objektů, každý může vypisovat jinam, mohou vypisovat zprávy různé důležitosti (např. všechny do souboru, na standardní výstup pouze některé),

Layout formát vypisované zprávy,

Filter umožňuje jemnější výběr zpráv, které se mají vypisovat (např. na základě porovnávání řetězců), vzhledem k velmi vzácnému použití nejsou filtry dále popisovány,

Obrázek č. 7 ukazuje jak jsou mezi sebou základní prostředky log4j provázány.



Obrázek č. 7 Provázání jednotlivých součástí log4j

Priorita

Log4j rozlišuje následující priority (konstanty jsou definovány ve třídě org.apache.log4j.Level) seřazené od nejnižší k nejvyšší:

úroveň (konstanta)	**metody objektu logger pro výpis**	**význam**
DEBUG	void debug(Object message) void debug(Object message, Throwable t)	zprávy, které jsou určené pro ladění aplikace
INFO	void info(Object message) void info(Object message, Throwable t)	zprávy o průběhu aplikace,
WARN	void warn(Object message) void warn(Object message, Throwable t)	zprávy upozorňující na možné problémy,
ERROR	void error(Object message) void error(Object message, Throwable t)	vážné chyby, které však umožňují další běh aplikace
FATAL	void fatal(Object message) void fatal(Object message, Throwable t)	chyby, které znamenají ukončení činnosti aplikace

tabulka 4 Úrovně priority definované v knihovně log4j

Objekt Logger může pro výpisy zpráv použít také následující obecné metody, ve kterých se specifikuje úroveň zprávy:

```
public void log(Level l, Object message);
public void log(Level l, Object message, Throwable t);
```

Druhý tvar s výjimkou (s objektem typu Throwable) umožňuje vypisovat standardní chybové hlášení.

Objektu Logger lze nastavit, od jaké úrovně priority má vypisovat zprávy. Např. lze nastavit, že má vypisovat od úrovně WARN, poté se zprávy s úrovní INFO a DEBUG nevypisují. Budou se vypisovat všechny zprávy úrovně WARN a úrovní vyšších (tj. úrovně ERROR a FATAL).

U každého objektu Appender lze také nastavit, od jaké priority má vypisovat – jeden Appender může vypisovat např. od úrovně INFO[4], druhý od úrovně ERROR.

[4] Objekt Logger předává zprávu instancím třídy Appender. Tj. pokud bude mít objekt Logger nastaveno vypisování od úrovně WARN, zprávy úrovně INFO se nebudou vypisovat.

Kategorie

Rozlišování zpráv podle priority je ve větších programech obvykle nedostatečné, proto je zaveden pojem kategorie, který umožňuje rozlišování zpráv dle dalšího kritéria. Kategorie mohou být definovány funkčně (např. kategorie security, database, database.select, ...) nebo v Javě častěji podle tříd. Kategorie vytvářejí hierarchickou strukturu obdobně jako třídy a balíčky – pro oddělování úrovní kategorii se používá znak tečka. Např. kategorie org.bar je následníkem kategorie org, kategorie org.bar.x je následníkem kategorie org.bar. Základem všech kategorii je nepojmenovaná kategorie root.

Při vytváření instance třídy Logger musíte zadat kategorii (nebo použít Logger kategorie root). Instance třídy Appender jsou vždy vázána na instanci třídy Logger určité kategorie.

Hierarchická struktura kategorii umožňuje „dědění“ vlastností z vyšších kategorii - pokud není přiřazena priorita k instanci třídy Logger dané kategorie, převezme se priorita od vyšší kategorie. V případě instancí tříd Appender se vypisuje pomocí všech instancí třídy Appender na této i na všech vyšších úrovních (pokud není stanoveno jinak).

Logger, Appender

Logger je nástroj pro programátora pro výpis zpráv. Přehled metod pro výpis je uveden v části o prioritách. Pro každou kategorii je vytvořena samostatná statická instance třídy Logger, kterou může programátor získat pomocí jedné z následujících továrních (factory) metod.

```
static Logger logger1 = Logger.getLogger("projekt.security");
```

instance logger1 odkazuje na Logger pro kategorii "projekt.security".

```
static Logger logger2 = Logger.getLogger(Pokus.class);
```

instance logger2 odkazuje na Logger pro kategorii vytvořenou na základě zařazení třídy Pokus do balíčků (pokud by byla v defaultním balíčku, poté by se kategorie jmenovala Pokus),

```
static Logger root = Logger.getRootLogger();
```

instance root odkazuje na Logger pro hlavní (root) kategorii,

Appender zajišťuje vlastní vypisování zpráv do souboru, na konzoli, do systémové logu atd. Inicializace se obvykle provádí přes konfigurační soubor včetně nastavení parametrů příslušné instance. V následující tabulce jsou uvedeny třídy, které implementují rozhraní Appender a slouží pro zápis logu na různá media apod.

třída	**popis**
ConsoleAppender	výpis na konzoli
FileAppender	výpis do souboru
DailyRollingFileAppender	výpis do souboru s rotací po určité době
RollingFileAppender	výpis do souboru s rotací
WriterAppender	výstup do Writer nebo do OutputStream
LF5Appender	výstup do grafického uživatelského rozhraní (SWING) se sofistikovaným rozhraním na zpracování zpráv,
NTEventLogAppender	výstup do Event Logu (systému logování ve Windows NT a následnících)
SyslogAppender	výstup do Syslogu (systém logování v Unixu),
JMSAppender	výstup do pošty ve formě JMS zpráv směrovaných do zadaného kanálu
SMTPAppender	výstup přes el. poštu,
SocketAppender	výstup na vzdálený počítač (používá se např. při RMI)
AsyncAppender	asynchronní výstup (tj. přes samostatné vlákno), na tento Appender jsou napojeny další, které zajišťují vlastní výstup,
JDBCAppender	výstup do databáze,
SNMPTrapAppender	generování událostí (SNMP Trap) pro SNMP konzoli,

tabulka 5 Přehled appenderů knihovny log4j

Layout

Layout určuje formát výstupu. Log4J obsahuje několik tříd pro formátování výstupu, nejčastěji se používají třídy:

- PatternLayout – formátuje zprávu obvykle na jeden řádek dle vzoru zadaného v konfiguraci,
- XMLLayout – výstup do souboru ve formátu XML,
- HTMLLayout – výstup do tabulky v HTML souboru, používá se např. při odesílání logu pomocí el. pošty,

PatternLayout

Výstup se řídí dle zadaného vzoru. Syntaxe pro zápis vzoru je odvozena od vzorů pro formátování výstupu v příkazu printf v jazyce C. Do vzoru je možné vložit jakýkoliv text a literály, které začínají znakem %. Místo každého literálu se před vypsáním dosadí příslušná hodnota. Vzor může vypadat následovně:

```
"%p [%t] %c - %m%n"
"%r [%t] %p %c %x - %m%n"
"%5p [%t] (%F:%L) - %m%n"
"%d{DATE} %5p %c - %m%n"
```

literál	význam
%c	kategorie
%C	jméno třídy (náročné na zjišťování)
%d	vypisuje datum a čas, log4j nabízí předpřipravené formáty %d{DATE}, %d{ISO8601} a %d{ABSOLUTE}, lze zadat vlastní formátování datumu a času
%F	jméno zdrojového souboru třídy (náročné na zjišťování)
%l	údaje o místu vyvolání (metoda, soubor, řádek), (časově náročné na zjišťování, ale užitečné)
%L	řádek zdrojového souboru, (náročné na zjišťování)
%m	vlastní zpráva
%M	jméno metody ze které voláno, (náročné na zjišťování)
%n	znak pro konec řádky (oddělovač řádek), doplní se správný dle operačního systému
%p	název priority
%r	počet milisekund od startu aplikace
%t	jméno vlákna
%x	NDC (nested diagnostic context) – používá se pro další rozlišování zpráv u vícevláknových či distribuovaných aplikací, může to být např. jméno uživatele, cookiies atd.
%X	MDC (mapped diagnosic context) – rozšíření NDC na více položek
%%	vypsání znaku procenta

tabulka 6 Význam formátovacích znaků pro třídu PatternLayuot

Literály lze doplnit o číslo určující délku – např. %5p znamená minimální délku 5 zarovnáno vpravo, %-20.30 znamená minimální délku 20 znaků se zarovnáním vlevo a maximální délku 30.

Konfigurace log4j - příklady

Log4j je možné konfigurovat v programu nebo pomocí konfiguračního souboru. Výrazně je preferována konfigurace pomocí konfiguračního souboru, neboť je mnohem více flexibilní. Log4j rozeznává dva formáty konfiguračního souboru – formát properties a xml formát.

Při inicializaci log4j v programu se hledá soubor s konfigurací podle nastavení system property **log4j.configuration**, ve které by měl být uveden soubor či URL, na kterém lze nalézt konfigurační soubor. Pokud jméno souboru (URL) končí znaky xml, předpokládá se, že je ve formátu xml, v opačném případě se předpokládá formát properties. Pokud není nastavena systémová proměnná log4j.configuration, zkouší se najít v aktuální adresáři soubor **log4j.properties**, popř. **log4j.xml**.

Jednoduchý příklad

Následující konfigurační soubor obsahuje přesměrování zpráv pro kategorii root na konzoli (ConsoleAppender) s použitím formátu PatternLayout s předpisem "%p [%t] %c - %m%n ":

```
# Pro root kategorii se nastaví priorita DEBUG a appender se jménem
A1
log4j.rootCategory=DEBUG, A1

# ke jménu A1 se přiřadí ConsoleAppender (výpis na konzoli).
log4j.appender.A1=org.apache.log4j.ConsoleAppender
```

```
# A1 používá pro formátování PatternLayout s dále uvedeným vzorem
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%p [%t] %c - %m%n
```

Následující program tento konfigurační soubor (jmenuje se log4j.demo1.properties) využije:

```
import org.apache.log4j.*;
public class Demo1 {
   static Logger logger = Logger.getLogger(Demo1.class);

   public static void main (String [] args) {
      logger.debug("Message 1");
      logger.warn("Message 2");
   }
}
```

Po spuštění příkazem

```
java -cp log4j.jar;. -Dlog4j.configuration=log4j.demo1.properties
Demo1
```

bude výpis následující:

```
DEBUG [main] Demo1 – Message 1
WARN [main] Demo1 – Message 2
```

Složitější příklad konfiguračního souboru

Následuje příklad složitějšího konfiguračního souboru pro aplikaci, která má dvě kategorie: imap a quota. Na konzoli se mají zobrazovat zprávy od priority INFO, do souboru quota.log (který má rotovat) se mají zapisovat zprávy od priority WARN.

```
# Root kategorie: výpis od priority INFO appendery se jménem stdout
a R
log4j.rootLogger=INFO, stdout, R

# Nastavení appenderu stdout – ConsoleAppender, PatternLayout a vzor
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p %c - %m%n

# Nastavní appenderu R – RollingFileAppender do souboru s rotováním
log4j.appender.R=org.apache.log4j.RollingFileAppender
# jméno souboru
log4j.appender.R.File=quota.log
# maximální velikost souboru
log4j.appender.R.MaxFileSize=100KB
# počet záloh nastaven na 5
log4j.appender.R.MaxBackupIndex=5
# od jaké priority (úrovně) se bude vypisovat
log4j.appender.R.Threshold=WARN
# pro výpis se použije PatternLayout s příslušným vzorem
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{DATE} %5p %c - %m%n
```

Programátor má podezření na chybu a proto chce pro kategorii imap vypisovat všechny zprávy (tj včetně priority debug) do souboru debug-imap.log. Nemusí zasahovat do zdrojového kódu, postačí doplnit na konec předchozího konfiguračního souboru následující řádky:

```
# Pro kategorii imap je prioriat DEBUG a appender se jménem D
log4j.logger.imap=DEBUG,D
# appender D je typu FileAppender, vypisuje do souboru debug.log
log4j.appender.D=org.apache.log4j.FileAppender
log4j.appender.D.File=debug.log
# nesmí se zapomenout na přiřazení formátování zpráv
log4j.appender.D.layout=org.apache.log4j.PatternLayout
log4j.appender.D.layout.ConversionPattern=%d{DATE} %5p %c - %m%n
```

Zprávy priority DEBUG z kategorie IMAP se též vypisují na obrazovku, neboť appendery se skládají dohromady v rámci hierarchické struktury kategorii. Do souboru quota.log se DEBUG zprávy nevypisují, neboť appender R má nastavenu minimální prioritu na úrovni WARN. Výpisu DEBUG zpráv na obrazovku lze nejsnáze zabránit nastavením minimální priority u appenderu stdout.

Použití třídy BasicConfigurator

Základní konfiguraci je možné nastavit v programu pomocí statické metody BasicConfigurator.configure(), která nastaví ConsoleAppender pro kategorii root. Tento Appender použije pro formátování výstupu PatternLayout s tímto formátem: "%r [%t] %p %c %x - %m%n".

Následující příklad ukazuje použití tohoto způsobu konfigurace. V příkladu lze vidět nastavení minimální priority pro Logger i způsob, jak vypsat údaje o výjimce.

```
import org.apache.log4j.*;

public class Demo {
    static Logger log = Logger.getLogger(Demo.class.getName());

    public static void main(String[] args) {
        BasicConfigurator.configure();
        log.setLevel(Level.WARN);
        log.info("Zaciname...");
        log.debug("Tato zprava nebude vypsana!");
        try {
            //Deleni nulou
            int x = 5;
            int y = 20 / (5 - x);
        }
        catch (Exception e) {
            log.error("Chyba!", e);
        }
    }
}
```

Výstup z toho programu bude následující:

```
0 [main] INFO Demo  - Zaciname...
110 [main] ERROR Demo  - Chyba!
java.lang.ArithmeticException: / by zero
        at Demo.main(Demo.java:14)
```

Konfigurační soubor jako parametr příkazové řádky

V tomto příkladě si ukážeme, jak by mohla v programu vypadat obsluha inicializace log4j pomocí zadání jména konfiguračního souboru jako parametru na příkazovém řádku.
V příkladu je také ukázka konfigurace log4j přímo v programu (pro případ, že není zadán

konfigurační soubor). Výpis ukazuje jaké importy jsou pro aplikaci třeba a část metody main, která slouží pro inicializaci logování.

```
import org.apache.log4j.*;
import org.apache.log4j.xml.DOMConfigurator;
……..
……..
    if(args.length == 0) {
        Logger root = Logger.getRootLogger();
        Layout layout = new PatternLayout
                          ("%p [%t] %c (%F:%L) - %m%n");
        root.addAppender(new ConsoleAppender
                     (layout,ConsoleAppender.SYSTEM_OUT));
    }
    else {
        if(args.length == 1) {
            if(args[0].endsWith("xml")) {
                DOMConfigurator.configure(args[0]);
            }
            else {
                PropertyConfigurator.configure(args[0]);
            }
        };
    }
```

Test existence instance Appender

Následující část kódu ukazuje, jak zjistit, zda je pro konkrétní Logger přiřazen Appender (pokud není přiřazen žádný Appender (ani se nedědí), tak se vypisuje na System.err chybové hlášení při volání metody pro výpis).

```
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

    static Logger logger = Logger.getLogger(Pokus1.class);

    public static void main (String args []) {
        if (logger.getAllAppenders() ==
            org.apache.log4j.helpers.NullEnumeration.getInstance())
        {
            BasicConfigurator.configure();
        };
```

XML konfigurační soubor

V XML konfiguračních souborech lze definovat stejné parametry jako v konfiguračních souborech ve formátu properties. Navíc lze konfigurovat některé rozšiřující funkce jako jsou filtry, obsluha chyb log4j či provázané instance třídy Appender (používá se u třídy AsyncAppender). Následující konfigurační soubor odpovídá první ukázce jednoduchého konfiguračního souboru – souboru log4j.demo1.properties.

```
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>

    <appender name="A1" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                           value="%p [%t] %c - %m%n"/>
        </layout>
    </appender>

    <root>
        <priority value ="debug" />
        <appender-ref ref="A1" />
    </root>

</log4j:configuration>
```

Logování v JDK 1.4

Firma Sun do verze 1.4 J2SDK vložila knihovnu pro podporu logování. Potřebné třídy jsou dostupné v balíčku java.util.logging. Koncepce je velmi podobná koncepci log4j, používají se trochu jiné pojmy:

pojem v log4j	**pojem v J2SDK 1.4**	**rozdíly**
Logger	Logger	není významný rozdíl,
Appender	Handler	v JDK 1.4 je k dispozici méně možností: ConsoleHandler, StreamHandler, FileHandler, SocketHandler a MemoryHandler,
Layout	Formatter	v JDK 1.4 je k dispozici pouze SimpleFormatter a XMLFormatter
priority	log level	v JDK 1.4 se rozeznává 7 úrovní (seřazeny od nejméně významné): FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE – těmto názvům odpovídají příslušné metody třídy Logger pro výpis zpráv (pro variantu s výjimkou se musí použít obecná metoda log),
category	namespace	hierarchické stejně jako v log4j,
konfigurační soubor	konfigurační soubor	v JDK 1.4 soubor ve formátu properties, implicitně se načítá soubor lib/logging.properties z JRE adresáře, soubor lze zadat přes systémovou proměnnou java.util.logging.config.file (např. z příkazové řádky: -Djava.util.logging.config.file=my.log.config),

tabulka 7 Porovnání používaných pojmů v log4j a J2SDK 1.4

Následuje příklad programu, který používá logování z JDK 1.4

```
import java.util.logging.*;

public class DemoJDK{
    // získání instance třídy Logger.
    private static Logger logger = Logger.getLogger("DemoJDK");

    public static void main(String argv[]){
        // výpis zprávy na úrovni FINE
        logger.fine("doing stuff");
        try{
            // nějaký kód ...
        }
        catch (Error ex){
            // výpis chyby na úrovni WARNING
            logger.log(Level.WARNING,"problem pri zpracovani",ex);
        }
        logger.fine("done");
    }
}
```

Konfigurační soubor pro logování v JDK 1.4 (jre/lib/logging.properties) vypisuje zprávy na konzolu od úrovně INFO. Vypadá následovně (vynechána část komentářů):

```
############################################################
#   Global properties
############################################################

# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during VM startup.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.

handlers= java.util.logging.ConsoleHandler

# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers. For any given facility this global level
# can be overriden by a facility specific level
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.level= INFO
# Limit the message that are printed on the console to INFO and
above.

java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter

############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################

# For example, set the com.xyz.foo logger to only log SEVERE
# messages:
com.xyz.foo.level = SEVERE
```

Některé aplikace používají knihovnu commons-logging ze serveru jakarta.apache.org, která skrývá jednotlivé implementace logování v Javě a vytváří pro ně obecné rozhraní (uživatelé těchto aplikací si poté mohou vybrat, jakou implementaci logování zvolí). Příkladem takové aplikace je webový kontejner Tomcat.