Logging i Java

Denne artikel beskriver baggrunden for logging frameworks og viser hvordan man bruger 2 af de mest almindelige: Apache Log4J og J2SE/Java SE 1.4 logging.

Den forudsætter kendskab til Java og noget generel udviklings erfaring.

Skrevet den **18. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

Historie:
V1.0 - 19/01/2004 - original
V1.1 - 31/01/2004 - forbedret formatering
V1.2 - 09/02/2004 - flere ændringer af formatering
V1.3 - 18/02/2010 - smårettelser

Indledning

Alle programmører kender til troubleshooting af mystiske fejl.

Og på trods af online debuggere har vundet meget frem de sidste 20 år, så er der stadig et stort behov for noget log output.

Lad os starte med et simpelt eksempel af hvordan et logging approach typisk udvikler sig.

Vi tager udgangspunkt i koden:

```
public class Log1 {
    public static int fac(int n) {
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        return res;
    }
    public static void main(String[] args) {
        System.out.println(fac(50));
    }
}
```

som på "mystisk" vis skriver 0 ud.

(den erfarne programmør har selvfølgelig gennemskuet at

det er et overflow problem, men vi fortsætter med eksemplet)

Vi sætter en System.err.println ind:

```
public class Log1 {
    public static int fac(int n) {
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        System.err.println("fac : n = " + n + " res = " + res);
    return res;
    }
    public static void main(String[] args) {
        System.out.println(fac(50));
    }
}
```

Og output viser hurtigt fejlen:

```
fac : n = 1 res = 1
fac : n = 2 res = 2
fac : n = 3 res = 6
fac : n = 4 res = 24
fac : n = 5 res = 120
fac : n = 6 res = 720
fac : n = 7 res = 5040
fac : n = 8 res = 40320
fac : n = 9 res = 362880
fac : n = 10 res = 3628800
fac : n = 11 res = 39916800
fac : n = 12 res = 479001600
fac : n = 13 res = 1932053504
fac : n = 14 res = 1278945280 <----------
fac : n = 15 res = 2004310016
fac : n = 16 res = 2004189184
fac : n = 17 res = -288522240
fac : n = 18 res = -898433024
fac : n = 19 res = 109641728
fac : n = 20 res = -2102132736
...
```

Koden fejler når n > 13.

Så sletter man System.err.println igen.

Og næste uge har man igen et problem som gør at man sætter den samme linie ind igen.

Efter det er løst er man blevet lidt klogere og nøjes med at udkommentere linien i stedet for at slette den.

```
public class Log1 {
    public static int fac(int n) {
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        //System.err.println("fac : n = " + n + " res = " + res);
    return res;
    }
    public static void main(String[] args) {
        System.out.println(fac(50));
    }
}
```

Det fortsætter man så med i mange source filer.

Og efterhånden bliver det et større arbejde at fjerne udkommenteringer og builde hver gang man skal finde en fejl - og man begynder at glemme at udkommentere igen, hvilket giver en masse generende debug output for kollegerne.

Men så får man en genial ide.

```
import java.io.*;

public class Logging {
    public static boolean log = false;
    public static PrintStream dbg = System.err;
}
```

Og nu kan man kode som:

```
public class Log2 {
    public static int fac(int n) {
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        if(Logging.log) Logging.dbg.println("fac : n = " + n + " res = " +
```

```
res);
        return res;
    }
    public static void main(String[] args) {
        Logging.log = true;
        System.out.println(fac(50));
    }
}
```

Og enable/disable debug output med en enkelt linie i sit main program.

Det er også helt fint.

Indtil det kommer for dagens lys at samtlige af ens 24 kolleger også har deres lille logging klasse. Og at både konfiguration og output er totalt inkompatible.

Og nu er man så klar til et logging framework, som alle *skal* bruge.

De 2 mest gængse logging frameworks i Java verdenen er Log4J og J2SE/Java SE 1.4 logging.

Log4J

Installationen er simpel: man downloader fra http://logging.apache.org/log4j/, unzipper filen og putter jar filen i classpath.

Man bruger Log4J ved at lave en klasse variabel som:

```
private final static Logger logger = Logger.getLogger("loggernavn");
```

og kalde:

```
logger.debug("Dette er en debug sætning");
logger.info("Dette er en informativ sætning");
logger.warn("Dette er en advarsels sætning");
logger.error("Dette er en fejl sætning");
logger.fatal("Dette er en fatal fejl sætning");
```

Note: per konvention bruger man ofte pakke.Klasse som logger navn (det er praktisk da loggere konfigureres hirakisk).

I vores lille eksempel bliver det til:

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

public class Log3 {
    private final static Logger logger = Logger.getLogger("Log3");
    public static int fac(int n) {
```

```
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        logger.debug("fac : n = " + n + " res = " + res);
        return res;
    }
    public static void main(String[] args) {
        logger.info("Start");
        System.out.println(fac(50));
        logger.info("End");
    }
}
```

Så kører man programmet med enten:

```
-Dlog4j.configuration=file:///C:/log4j.properties
```

eller:

```
-Dlog4j.configuration=file:///C:/log4j.xml
```

hvor log4j.properties ser ud som:

```
# Log3 logger: minimum level=debug, two appenders (logfile + console)
log4j.category.Log3 = debug, logfile, console
# console: minimum level=info, special format
log4j.appender.console.threshold = info
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout = org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern = %-30c %d %-5p %m%n
# logfile: minmum level=debug, fil=log4.log, special format
log4j.appender.logfile.threshold = debug
log4j.appender.logfile = org.apache.log4j.FileAppender
log4j.appender.logfile.file = C:\log4j.log
log4j.appender.logfile.layout = org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern = %-30c %d %-5p %m%n
```

og log4j.xml ser ud som:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
debug="true">
```

```
    <!--
      console: minimum level=info, special format
    -->
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="threshold" value="info"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-30c %d %-5p %m%n"/>
        </layout>
    </appender>
    <!--
      logfile: minmum level=debug, fil=log4.log, special format
    -->
    <appender name="logfile" class="org.apache.log4j.FileAppender">
        <param name="threshold" value="debug"/>
        <param name="file" value="C:\log4j.log"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-30c %d %-5p %m%n"/>
        </layout>
    </appender>
    <!--
      Log3 logger: minimum level=debug, two appenders (logfile + console)
    -->
    <logger name="Log3">
        <level value="debug"/>
        <appender-ref ref="console"/>
        <appender-ref ref="logfile"/>
    </logger>
</log4j:configuration>
```

så får man:

```
Log3                           2004-01-19 22:17:24,000 INFO  Start
0
Log3                           2004-01-19 22:17:24,015 INFO  End
```

til skærmen og:

```
Log3                           2004-01-19 22:17:24,000 INFO  Start
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 1 res = 1
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 2 res = 2
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 3 res = 6
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 4 res =
24
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 5 res =
120
Log3                           2004-01-19 22:17:24,000 DEBUG fac : n = 6 res =
720
Log3                           2004-01-19 22:17:24,015 DEBUG fac : n = 7 res =
5040
Log3                           2004-01-19 22:17:24,015 DEBUG fac : n = 8 res =
40320
Log3                           2004-01-19 22:17:24,015 DEBUG fac : n = 9 res =
```

```
362880
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 10 res =
3628800
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 11 res =
39916800
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 12 res =
479001600
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 13 res =
1932053504
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 14 res =
1278945280
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 15 res =
2004310016
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 16 res =
2004189184
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 17 res =
-288522240
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 18 res =
-898433024
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 19 res =
109641728
Log3                            2004-01-19 22:17:24,015 DEBUG fac : n = 20 res =
-2102132736
...
Log3                            2004-01-19 22:17:24,015 INFO  End
```

til log4j.log !

Note: den hirakiske konfiguration betyder at hvis man konfigurerer logger xxx
så påvirker det automatisk xxx.aaa og xxx.bbb !

J2SE/Java SE 1.4 logging
Denne logger er indbygget i alle Java 1.4 og nyere.

Nogen gange kaldes den JUL (efter java.util.logging pakken).

Man bruger den ved at lave en klasse variabel som:

```
private final static Logger logger = Logger.getLogger("loggernavn");
```

og kalde:

```
logger.finest("Dette er en debug level 3 sætning");
logger.finer("Dette er en debug level 2 sætning");
logger.fine("Dette er en debug level 1 sætning");
logger.config("Dette er en konfiguration sætning");
logger.info("Dette er en informativ sætning");
logger.warning("Dette er en advarsels sætning");
logger.severe("Dette er en alvorlig sætning");
```

Note: per konvention bruger man ofte pakke.Klasse som logger navn (det er praktisk
da loggere konfigureres hirakisk).

I vores lille eksempel bliver det til:

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

public class Log4 {
    private final static Logger logger = Logger.getLogger("Log4");
    public static int fac(int n) {
        int res;
        if (n <= 1) {
            res = 1;
        } else {
            res = n * fac(n - 1);
        }
        logger.fine("fac : n = " + n + " res = " + res);
        return res;
    }
    public static void main(String[] args) {
        logger.info("Start");
        System.out.println(fac(50));
        logger.info("End");
    }
}
```

Så kører man programmet med:

```
-Djava.util.logging.config.file=C:\log.properties
```

hvor log.properties ser ud som:

```
# Log4 logger: minimum level=fine, two handlers (console + logfile)
Log4.level = FINE
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler
# console: minimum level=info
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
# logfile: minmum level=fine, fil=log.log
java.util.logging.FileHandler.level = FINE
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.pattern = C:\log.log
```

så får man:

```
Jan 19, 2004 10:52:26 PM Log4 main
INFO: Start
0
Jan 19, 2004 10:52:26 PM Log4 main
INFO: End
```

til skærmen og:

```
Jan 19, 2004 10:52:26 PM Log4 main
INFO: Start
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 1 res = 1
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 2 res = 2
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 3 res = 6
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 4 res = 24
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 5 res = 120
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 6 res = 720
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 7 res = 5040
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 8 res = 40320
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 9 res = 362880
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 10 res = 3628800
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 11 res = 39916800
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 12 res = 479001600
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 13 res = 1932053504
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 14 res = 1278945280
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 15 res = 2004310016
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 16 res = 2004189184
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 17 res = -288522240
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 18 res = -898433024
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 19 res = 109641728
Jan 19, 2004 10:52:26 PM Log4 fac
FINE: fac : n = 20 res = -2102132736
...
Jan 19, 2004 10:52:26 PM Log4 main
INFO: End
```

til log.log !

Note: den hirakiske konfiguration betyder at hvis man konfigurerer logger xxx
så påvirker det automatisk xxx.aaa og xxx.bbb !

Log4J eller J2SE/Java SE 1.4 logging

Den opmærksomme læser har nu konkluderet at SUN har lånt rigtigt meget fra Apache Log4J !!

Så hvad vælger man ?

Client applikation - helt klart J2SE/Java SE 1.4 logging fordi den har det der skal bruges og den er der uden man skal installere noget.

Server applikation - her må man afgøre om man kan nøjes med J2SE/Java SE 1.4 logging eller man skal have fat på Apache Log4J.

Apache Log4J indeholder bl.a. en del gode appendere til server brug:
- org.apache.log4j.DailyRollingFileAppender
- org.apache.log4j.RollingFileAppender
- org.apache.log4j.jdbc.JDBCAppender
- org.apache.log4j.net.JMSAppender
- org.apache.log4j.net.SMTPAppender
- org.apache.log4j.net.SyslogAppender
- org.apache.log4j.nt.NTEventLogAppender

Apache har også lavet et tredie logging framework Commons Logging, (http://jakarta.apache.org/commons/logging.html), hvor valget mellem Log4J og J2SE/Java SE 1.4 logging er konfigurerbart.

Kommentar af simonvalter d. 20. Jan 2004 | 1

Smart! og lige til at gå til efter at have læst denne artikel.

Kommentar af dodger d. 13. Oct 2005 | 2

Kommentar af margitbork d. 24. Mar 2004 | 3