Common conversion patterns for log4j's PatternLayout

Last Updated on 22 July 2015 | Print Email

You will find in this article some useful conversion patterns compiled for daily use of log4j as a preferred logging mechanism in Java development.

To recall, a conversion pattern specifies how log messages are formatted, using a combination of literals, conversion characters and format modifiers. Consider the following pattern:

```
log4j.appender.ConsoleAppender.layout.ConversionPattern=[%-5p] %d %c - %m%n
```

Where:

- The percent sign (%) is a prefix for any conversion characters.
- Conversion characters are: **p, d, c, m** and **n**. These characters are explained in the table below.
- **-5p** is the format modifier for the conversion character p (priority), which left justifies the priority name with minimum 5 characters.
- Other characters are literals: **[,], -** and spaces

That pattern will format log messages something like this:

```
1  [DEBUG] 2012-11-02 21:57:53,661 MyClass - this is a debug log message
2  [INFO ] 2012-11-02 21:57:53,662 MyClass - this is a information log message
3  [WARN ] 2012-11-02 21:57:53,662 MyClass - this is a warning log message
```

The following table briefly summarizes the conversion characters defined by log4j:

What to print	Conversion character	Performance
Category name (or logger name)	**c**	Fast
Fully qualified class name	**C**	Slow
Date and time	**d** **d{format}**	*Slow if using JDK's formatters.* *Fast if using log4j's formatters.*
File name of Java class	**F**	**Extremely slow**
Location (class, method and line number)	**l**	**Extremely slow**
Line number only	**L**	**Extremely slow**
Log message	**m**	Fast
Method name	**M**	**Extremely slow**
Priority (level)	**p**	Fast
New line separator	**n**	Fast
Thread name	**t**	Fast
Time elapsed (milliseconds)	**r**	Fast
Thread's nested diagnostic context	**x**	Fast
Thread's mapped diagnostic context	**X**	Fast
Percent sign	**%%**	Fast

And next are some useful patterns compiled by us for common usage and specific needs. The example output is based on this small program (assuming log4j is initialized elsewhere. See the article: How to configure log4j as logging mechanism in Java):

```
1  import java.io.File;
2  import org.apache.log4j.Logger;
3  import org.apache.log4j.PropertyConfigurator;
4
5  public class MyClass {
6
7      static Logger logger = Logger.getLogger(MyClass.class);
8
9      void foo() {
10         logger.info("this is a log message");
11     }
12 }
```

Simple conversion pattern

Perhaps the simplest pattern contains only the log message, but that is not informative for debugging. So a simple pattern would contain priority, category, method name, and message.

Pattern: **[%p] %c %M - %m%n**

Example output:

```
[INFO] MyClass - foo - this is a log message
```

NOTE: the **%n** should be used to add new line character at the end of every message.

Standard conversion pattern

A standard (and probably most used) pattern would contain the following information: priority, date and time, category, method, and message.

Pattern: **[%p] %d %c %M - %m%n**

Example output:

```
[INFO] 2012-11-02 22:46:43,896 MyClass foo - this is a log message
```

Conversion patterns where date and time matters

Sometimes it is critical to measure time in the log output. The **%d** conversion character outputs date and time in ISO8601 format by default. However we can use a date format specifier to format the date and time explicitly, for example:

%d{MM-dd-yyyy HH:mm:ss,SSS}

If you need time granularity only to second:

- Pattern: **[%p] %d{MM-dd-yyyy HH:mm:ss} %c %M - %m%n**
- Example output:

```
[INFO] 11-02-2012 23:08:10 MyClass foo - this is a log message
```

If you need time granularity to milliseconds:

- Pattern: **[%p] %d{MM-dd-yyyy HH:mm:ss,SSS} %c %M - %m%n**
- Example output:

```
[INFO] 11-02-2012 23:09:38,304 MyClass foo - this is a log message
```

Using ISO8601 date format specifier:

- Pattern: **[%p] %d{ISO8601} %c %M - %m%n**
- Example output:

```
[INFO] 2012-11-02 23:13:58,300 MyClass foo - this is a log message
```

Using ABSOLUTE date format specifier:

- Pattern: **[%p] %d{ABSOLUTE} %c %M - %m%n**
- Example output:

```
[INFO] 23:17:07,097 MyClass foo - this is a log message
```

Using DATE date format specifier:

- Pattern: **[%p] %d{DATE} %c %M - %m%n**
- Example output:

```
[INFO] 02 Nov 2012 23:18:19,547 MyClass foo - this is a log message
```

Conversion patterns where thread context matters

For debugging multi-threaded applications, log4j provides the following conversion characters:

- **%t**: thread name
- **%x**: thread's nested diagnostic context.
- **%X**: thread's mapped diagnostic context.

Pattern: **[%p] %d [%t] %x %c %M - %m%n**

Example output:

```
[INFO] 2012-11-02 23:28:26,178 [main] MyClass foo - this is a log message
```

Share this article: