1 Logging with log4j

This tutorial explains how to set up log4j with email, files and stdout. It compares XML to properties configuration files, shows how to change LogLevels for a running application. Furthermore, we explain best practices on logging and exception handling.

2 General

Author: Sebastian Hennebrueder

Date: February, 22th, 2007

Used software and frameworks

Tomcat 5.5

Log4j 1.2.14

PDF version of the tutorial:

http://www.laliluna.de/download/log4j-tutorial-en.pdf

Source code

http://www.laliluna.de/download/log4j-tutorial.zip

3 Beginning

You can download the current version of log4j from the project home page.

http://logging.apache.org/log4j/

Make sure that you use use md5sum to check that the downloaded file is not hacked.

In a linux console you can type the following and compare the number to that from the home page:

```
md5sum logging-log4j-1.2.14.zip
```

There are md5sum tools for windows as well. For Firefox you can install the md hash tool extension and check directly from the download windows.

4 First example

4.1 log4j.properties example

Create a Java project.

Add the log4j.jar to the build path of the project.

Create a file named log4j.properties in the src folder with the following content.

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n
log4j.rootLogger=debug, stdout
```

Create a class with the following content:

```
package de.laliluna.logexample;
import org.apache.log4j.Logger;

public class LogClass {
    private static org.apache.log4j.Logger log = Logger
     .getLogger(LogClass.class);

    public static void main(String[] args) {

    log.trace("Trace");
    log.debug("Debug");
    log.info("Info");
    log.warn("Warn");
    log.error("Error");
    log.fatal("Fatal");

    }
}
```

Run it. You should see the log messages in the console.

```
08:50:49,661 DEBUG LogClass:29 - Debug
08:50:49,663  INFO LogClass:30 - Info
08:50:49,663  WARN LogClass:31 - Warn
08:50:49,663 ERROR LogClass:32 - Error
08:50:49,664 FATAL LogClass:33 - Fatal
```

Change the line

```
log4j.rootLogger=debug, stdout
```

to

```
log4j.rootLogger=warn, stdout
```

and run your java application again.

What did we learn?

- Log4j does look for a file named log4j.properties in the src folder.
- We get a Logger by calling Logger.getLogger
- Do not use Category.getCategory to get a logger. This is deprecated.
- You can influence what is logged by setting the log level.
- How to log messages with the following levels: trace, debug, info, warn, error and fatal

4.2 log4j.xml example

Create a file named log4j.xml with the following content in your src folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration>
 <appender name="stdout" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.PatternLayout">
     <param name="ConversionPattern" value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n"/>
   </layout>
 </appender>
 <root>
   <priority value="debug"></priority>
   <appender-ref ref="stdout"/>
 </root>
</log4j:configuration>
```

Copy the **log4j.dtd** into the source folder as well. You can find it in the download of log4j. The XML requires a dom4j.jar which might not be included in older Java versions. You do not need it with Java 5

You can test your configuration the same way as the former example.

4.3 Log level

The following Levels are available. But you can define custom levels as well. Examples are provided with the log4j download.

Level	Description
all	All levels including custom levels
trace (since log4j 1.2.12)	developing only, can be used to follow the program execution.
debug	developing only, for debugging purpose
info	Production optionally, Course grained (rarely written informations), I use it to print that a configuration is initialized, a long running import job is starting and ending.
warn	Production, simple application error or unexpected behaviour. Application can continue. I warn for example in case of bad login attemps, unexpected data during import jobs
error	Production, application error/exception but application can continue. Part of the application is probably not working.
fatal	Production, fatal application error/exception, application cannot continue, for example database is down.
no	Do not log at all.

5 Log4j configuration

5.1 Layout of the log file

The layout specifies how a log message looks like.

First you define the layout.

```
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
```

The pattern layout requires another parameter, i.e. the pattern.

```
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n
```

The best up-to-date documentation about available layouts can be found in the API documentation:

http://logging.apache.org/log4j/docs/api/org/apache/log4j/Layout.html

There you can see that we have DateLayout, HTMLLayout, PatternLayout, SimpleLayout, XMLLayout as options.

SimpleLayout has no properties to be set. It is simple.

We used PatternLayout in our example and we set a property named ConversionPattern. This property allows us to define the log output.

`%d{ABSOLUTE}`	Date in Format Absolute
`%5p`	%5 defines a right justified print with 5 characters, p prints the priority of the log message
`%c{1}:%L - %m%n`	And the other settings. Very simple. They are all explained in the API.

The options to influence the layout are explained perfectly in the API documentation:

http://logging.apache.org/log4j/docs/api/org/apache/log4j/PatternLayout.html

5.2 Custom Layout

If the configuration options does not suite your needs, you can define custom layouts as well. Examples for custom layout are provided with the log4j download. Have a look in the examples directory.

5.3 Types of log appender

An appender specifies where your log messages are written to. There is a wide choice of appenders available. All appenders are direct or indirect subclasses of the AppenderSkeleton. Therefore we can find all options on the following API page:

http://logging.apache.org/log4j/docs/api/org/apache/log4j/AppenderSkeleton.html

The console and the file appender are a subclass of WriterAppender.

Later on, we are going to choose examples for the following appenders.

ConsoleAppender	Logs to console
FileAppender	Logs to a file
SMTPAppender	Logs by email
RollingFileAppender	Logs to a file, starts a new file once the max size is reached. (An alternative is the DailyRollingFileAppender which creates on file

	per day)

But there are as well:

AsyncAppender, JDBCAppender, JMSAppender, LF5Appender, NTEventLogAppender, NullAppender, NullAppender, SMTPAppender, SocketAppender, SocketHubAppender, SyslogAppender, TelnetAppender, DailyRollingFileAppender, RollingFileAppender.

Custom appenders can be created as well. The log4j download comes with a whole bunch of samples in the examples directory.

5.4 log4j.xml versus log4j.properties

Properties can be defined by a properties file or by an XML file. Log4j looks for a file named log4j.xml and then for a file named log4j.properties. Both must be placed in the src folder.

The property file is less verbose than an XML file. The XML requires the **log4j.dtd** to be placed in the source folder as well. The XML requires a dom4j.jar which might not be included in older Java versions.

The properties file does not support some advanced configuration options like Filters, custom ErrorHandlers and a special type of appenders, i.e. AsyncAppender. ErrorHandlers defines how errors in log4j itself are handled, for example badly configured appenders. Filters are more interesting. From the available filters, I think that the level range filter is really missing for property files.

This filter allows to define that a appender should receive log messages from Level INFO to WARN. This allows to split log messages across different logfiles. One for DEBUGGING messages, another for warnings, ...

The property appender only supports a minimum level. If you set it do INFO, you will receive WARN, ERROR and FATAL messages as well.

Here are two logfiles examples for a simple configuration:

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.rootLogger=debug, stdout
```

and

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration>
 <appender name="stdout" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.SimpleLayout"></layout>
 </appender>
 <root>
   <priority value="debug"></priority>
   <appender-ref ref="stdout"/>
 </root>
</log4j:configuration>
```

5.5 Loading the configuration

Log4j will first check for a file log4j.xml and then for a log4j.properties file in the root directory of the classes folder (= src folder before compilation).

You can load other configurations as well. Here are some examples:

```
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.helpers.Loader;
import org.apache.log4j.xml.DOMConfigurator;

.............. snip ...........

// use the loader helper from log4j
URL url = Loader.getResource("my.properties");
PropertyConfigurator.configure(url);

 // use the same class loader as your class
URL url = LogClass.class.getResource("/my.properties");
PropertyConfigurator.configure(url);

 // load custom XML configuration
URL url = Loader.getResource("my.xml");
DOMConfigurator.configure(url);
```

In a web application you might configure a servlet to be loaded on startup to initialize your configuration.

Keep in mind that this is not required, if you use the default names and folders for the configuration file.

5.5 Reconfigure a running log4j configuration

If you analyse a problem you frequently want to change the log level of a running application server. This chapter explains how you can do this. I used Tomcat as example server but you can use any application server you like.

The XML actually offers a method to watch changes in config files.

http://logging.apache.org/log4j/docs/api/org/apache/log4j/xml/DOMConfigurator.html#configureAndWatch(java.lang.String)

The problem is that it seems not to work in some situations. But this is no problem as it is quite easy to develop a short tool by yourself. We have two options. We could change the log level during runtime:

```
Logger root = Logger.getRootLogger();
root.setLevel(Level.WARN);
```

or we can reload the configuration:

```
// PropertyConfigurator.configure(url);
DOMConfigurator.configure(url);
```

The following example will check the configuration file in defined intervals and reconfigure log4j if any changes are found.

We need to create three things:

a) a monitor thread, monitoring the configuration file and reconfiguring log4j if needed

b) a servlet starting and stopping the monitor thread

c) an entry in the web.xml, to initialize the servlet

The following class monitors the logj4 configuration file and checks with the last change date has changed:

```
package de.laliluna.logexample;

import java.io.File;
```

```
import java.net.URL;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.xml.DOMConfigurator;

public class MonitorThread implements Runnable {

    private static Logger log = Logger.getLogger(MonitorThread.class);

    boolean interruped;

    private long checkIntervalMillis = 10000;

    private URL url;

    private File file;

    // stores the last modification time of the file
    private long lastModified = 0;

    public void run() {
System.out.println("Initialize " + url.getPath());
file = new File(url.getPath());
// PropertyConfigurator.configure(url);
DOMConfigurator.configure(url);
lastModified = file.lastModified();

monitor();
    }

    private void monitor() {
log.info("Starting log4j monitor");

while (!interruped) {

    // check if File changed
    long temp = file.lastModified();
    if (lastModified != temp) {
  log.info("Initialize log4j configuration " + url.getPath());
  // PropertyConfigurator.configure(url);
  DOMConfigurator.configure(url);

  lastModified = temp;

    } else
  log.debug("Log4j configuration is not modified");
    try {
  Thread.currentThread().sleep(checkIntervalMillis);
    } catch (InterruptedException e) {
  interruped = true;
    }
}
log.info("Shutting down log4j monitor");
```

```
    }

    public URL getUrl() {
return url;
    }

    public void setUrl(URL url) {
this.url = url;
    }

    public long getCheckIntervalMillis() {
return checkIntervalMillis;
    }

    /**
         * Sets the interval for checking the url for changes. Unit is
         * milliseconds, 10000 = 10 seconds
         *
         * @param checkIntervalMillis
         */
    public void setCheckIntervalMillis(long checkIntervalMillis) {
this.checkIntervalMillis = checkIntervalMillis;
    }

    public boolean isInterruped() {
return interruped;
    }

    public void setInterruped(boolean interruped) {
this.interruped = interruped;
    }

}
```

The servlet starts and stops the monitor thread:

```
package de.laliluna.logexample;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

public class Log4jConfigLoader extends HttpServlet {

    private Thread thread;

  @Override
  public void destroy() {
    thread.interrupt();
    super.destroy();
   }

  public void init() throws ServletException {
    super.init();
    MonitorThread monitorThread = new MonitorThread();
    monitorThread.setCheckIntervalMillis(10000);
```

```
    monitorThread.setUrl(Log4jConfigLoader.class.getResource("/log4j.xml"));
    thread = new Thread(monitorThread);
    thread.start();
    }

}
```

We add the servlet to the web.xml to initialize it.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
  <servlet-name>log4j-init</servlet-name>
  <servlet-class>de.laliluna.logexample.Log4jConfigLoader</servlet-class>
  <load-on-startup>10</load-on-startup>
 </servlet>
</web-app>
```

6 Examples

6.1 Rolling File and errors to email

Log messages with Level info to fatal to a file and send messages from error to fatal by email. The file should be rolled every 100 KB.

You need mail.jar and activation.jar libraries from J2EE to send emails. Further properties of the SmtpAppender are described here:

http://logging.apache.org/log4j/docs/api/org/apache/log4j/net/SMTPAppender.html

log4j.properties

```
### file appender
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.maxFileSize=100KB
log4j.appender.file.maxBackupIndex=5
log4j.appender.file.File=test.log
log4j.appender.file.threshold=info
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

#email appender
log4j.appender.mail=org.apache.log4j.net.SMTPAppender
#defines how othen emails are send
log4j.appender.mail.BufferSize=1
log4j.appender.mail.SMTPHost="smtp.myservername.xx"
log4j.appender.mail.From=fromemail@myservername.xx
log4j.appender.mail.To=toemail@myservername.xx
log4j.appender.mail.Subject=Log ...
log4j.appender.mail.threshold=error
log4j.appender.mail.layout=org.apache.log4j.PatternLayout
log4j.appender.mail.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n
```

```
log4j.rootLogger=warn, file, mail
```

log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration>
 <appender name="file"
   class="org.apache.log4j.RollingFileAppender">
   <param name="maxFileSize" value="100KB" />
   <param name="maxBackupIndex" value="5" />
   <param name="File" value="test.log" />
   <param name="threshold" value="info"/>
   <layout class="org.apache.log4j.PatternLayout">
     <param name="ConversionPattern"
       value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n" />
   </layout>
 </appender>
 <appender name="mail" class="org.apache.log4j.net.SMTPAppender">
   <param name="SMTPHost" value="smtp.myservername.xx" />
   <param name="From" value="email@fromemail.xx" />
   <param name="To" value="toemail@toemail.xx" />
   <param name="Subject" value="[LOG] ..." />
   <param name="BufferSize" value="1" />
   <param name="threshold" value="error" />
   <layout class="org.apache.log4j.PatternLayout">
     <param name="ConversionPattern"
       value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n" />
   </layout>
 </appender>
 <root>
   <priority value="debug"></priority>
   <appender-ref ref="file" />
   <appender-ref ref="mail"/>
 </root>
</log4j:configuration>
 <root>
   <priority value="debug"></priority>
   <appender-ref ref="file" />
   <appender-ref ref="mail"/>
 </root>
</log4j:configuration>
```

6.2 Separate file

I want to have debugging messages to one file and other messages to another file. This can only be done with XML because we need a LevelRange filter.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration>
 <appender name="file"
   class="org.apache.log4j.RollingFileAppender">
   <param name="maxFileSize" value="100KB" />
```

```
  <param name="maxBackupIndex" value="5" />
  <param name="File" value="test.log" />
  <param name="threshold" value="info" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
      value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n" />
  </layout>
</appender>
<appender name="debugfile"
  class="org.apache.log4j.RollingFileAppender">
  <param name="maxFileSize" value="100KB" />
  <param name="maxBackupIndex" value="5" />
  <param name="File" value="debug.log" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
      value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n" />
  </layout>
  <filter class="org.apache.log4j.varia.LevelRangeFilter">
    <param name="LevelMin" value="debug" />
    <param name="LevelMax" value="debug" />
  </filter>
</appender>

<root>
  <priority value="debug"></priority>
  <appender-ref ref="debugfile" />
  <appender-ref ref="file" />
</root>
</log4j:configuration>
```

6.3 Other examples

The log4j download provides further examples as well.

7 Log4j and Tomcat

The configuration of log4j in tomcat is well described in the Tomcat documentation:
http://tomcat.apache.org/tomcat-5.5-doc/logging.html
Libraries are placed in common library directory. The configuration file for Tomcat is in common/classes directory. The configuration file for a application is placed in the WEB-INF/classes folder of the application.

There are some discussions that one application can steal others applications root logger, if it does not have its own log4j library in the WEB-INF-lib directory of the application. I tested a configuration with Tomcat 5.5 and two applications. Neither of the configuration was able to influence the others log4j configuration. As a consequence, it is sufficient if you place a logj4j library in the common/lib folder of your tomcat.

Hint: If you do not place a configuration file in one of your applications, it will use the Tomcat log files for logging.

Hint: If you do not want Tomcat to use log4j to log but only your application, you can place log4j in the WEB-INF-lib directory of your application as well.

8 Best practices for exception logging

I present some basic tips here. Further information can be found here:

http://today.java.net/lpt/a/280#throwingException
http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html

8.1 Do not use e.printStackTrace

e.printStackTrace prints to the console. You will only see this messages, if you have defined a console appender. If you use Tomcat or other application server with a service wrapper and define a console appender, you will blow up your wrapper.log.

```
try {
    ......... snip .......
} catch ( SomeException e) {
    e.printStackTrace();
}
```

You can use log.error(e,e). The second parameter passed an exception and will print the stack trace into the logfile.

```
try {
    ......... snip .......
} catch (SomeException e) {
    log.error("Exception Message", e);
    // display error message to customer
}
```

8.2 Don't log and throw again

```
try {
    ......... snip .......
} catch ( SomeException e) {
    log.error("Exception Message", e);
    throw e;
}
```

Do not catch an exception, log the stacktrace and then continue to throw it. If higher levels log a message as well, you will end up with a stacktrace printed 2 or more times into the log files.

8.3 Don't kill the stacktrace

```
    try{
... some code
    }catch(SQLException e){
        throw new RuntimeException("DB excpetion" +e.getMessage());
    }
```

This code will erase the stacktrace from the SQLException. This is not recommended, because you will loose important information about the exception. Better do the following.

```
 try{
... some code
    }catch(SQLException e){
        throw new RuntimeException("My Exception name", e);
    }
```

That's all for this tutorial.

9 Copyright and disclaimer