Edition
1

Date: 2002-05-26, 11:29:37 AM

SCOTT STARK

The JBoss Group

Using Log4j with JBoss

SCOTT STARK, AND THE JBOSS GROUP

Using Log4j with JBoss

© JBoss Group, LLC
2520 Sharondale Dr.
Atlanta, GA 30305 USA
sales@jbossgroup.com

Table of Content

PREFACE 2

ABOUT THE AUTHORS 2
ACKNOWLEDGMENTS 2

0. INTRODUCTION TO LOG4J 3

WHAT THIS BOOK COVERS 3

1. AN OVERVIEW OF THE LOG4J API 4

The org.apache.log4j.Category class 4
org.apache.log4j.Logger 6
The JBoss org.jboss.log.Logger wrapper 6
The org.apache.log4j.Appender interface 7
The org.apache.log4j.Layout class 8
Configuring log4j using org.apache.log4j.PropertyConfigurator 8
Configuring log4j using org.apache.log4j.xml.DOMConfigurator 10
Log4j usage patterns 18
The Log4jService MBean 19
SUMMARY 20

2. USING LOG4J 21

EMAIL NOTIFICATIONS BASED ON PRIORITY 21
The SMTPAppender 21
org.apache.log4j.AsyncAppender 23

3. INDEX 24

Table of Listings

Listing 1-1, A summary of the key methods in the log4j Category class. ____ 4
Listing 1-2, A summary of the key methods in the log4j 1.2 Logger class. ____ 6
Listing 1-3, The JBoss Logger class summary. ____ 6
Listing 1-4, Tthe standard JBoss-2.4.x log4j.properties configuration file. ____ 8
Listing 1-5, Tthe standard JBoss-3.0.x log4j.xml configuration file. ____ 14
Listing 2-1, A SMTPAppender configuration sample ____ 22
Listing 2-2, An AsyncAppender configuration sample that delegates messages to the ErrorNotifications SMTPAppender ____ 23

Table of Figures

Figure 1-1, The DTD for the configuration documents supported by the log4j version 1.1.3 DOMConfigurator. ________________________12
Figure 1-2, The DTD for the configuration documents supported by the log4j version 1.2.3 DOMConfigurator. ________________________14

Preface

Preface

About the Authors

<u>Scott Stark</u> , Ph.D., was born in Washington State of the U.S. in 1964. He started out as a chemical engineer and graduated with a B.S. from the University of Washington, and latter a PhD from the University of Delaware. While at Delaware it became apparent that computers and programming were to be his passion and so he made the study of applying massively parallel computers to difficult chemical engineering problems the subject of his PhD research. It has been all about distributed programming ever since. Scott currently serves as the Chief Technology Officer of the JBoss Group, LLC.

<u>JBoss Group LLC</u>, headed by Marc Fleury, is composed of over 1000 developers worldwide who are working to deliver a full range of J2EE tools, making JBoss the premier Enterprise Java application server for the Java 2 Enterprise Edition platform.

JBoss is an Open Source, standards-compliant, J2EE application server implemented in 100% Pure Java. The JBossServer and complement of products are delivered under the LGPL license. With over 1,000,000 downloads, JBoss is the most downloaded J2EE based server in the industry.

Acknowledgments

Thanks go out to the Log4j developers for establishing and maintaining such a flexible logging framework.

0. Introduction to Log4j

What is Log4j and why do we use it

Logging of messages is a common requirement in all applications. In a server environment it is a critical feature due to the distributed multi-user interaction that is characteristic of a server. Many users interact simultaneously with an application server and some degree of logging of the interactions is essential for support. A unique aspect of an application server is that many different developers may have contributed code to the applications that comprise the active components. The logging requirement could vary significantly between the various components or applications. What is needed is a flexible logging API that supports these use cases. The JBoss server has standardized on log4j as its logging API. The switch to log4j has been a gradual one, and as of the 2.4.4 release, log4j is the only logging API used internally by JBoss. JBoss-2.4.6 includes the log4j 1.1.3 release while JBoss-3.0.0 includes the log4j 1.2.3 release. The JBoss-2.4.6 version can be used with the log4j 1.2 release by simply replacing the log4j.jar in the JBoss distribution with the 1.2 version of the log4j.jar.

Although there are many logging APIs, including the JSR47 logging framework that is bundled with the current JDK 1.4 release, the log4j API appears to the most commonly used of all available. It is designed to be fast, flexible, and simple. Further, the log4j community is very active and responsive to both bug reports and feature requests. These are probably the most important criteria for an application server logging framework.

What this Book Covers

The focus of this book is using Log4j within JBoss. After an overview the Log4j API we go into the details of configuring and using Log4j. Both the log4j 1.1.3 log4j 1.2.3 releases are discussed as JBoss 2.4.x uses log4j 1.1.3 while JBoss 3.x uses log4j 1.2.3.

Chapter

1

1. An Overview of the Log4j API

What is the Log4j API and how does it fit into JBoss.

Log4j has four fundamental objects: categories, priorities, appenders and layouts. Of these, API users directly use only categories and maybe priorities. Together these components allow developers to log messages according to message type and priority, and to control at runtime how these messages are formatted and where they are reported. We will cover the basics of log4j to allow you to understand the JBoss log4j configuration and help get you started using log4j in your components. For additional documentation refer to the log4j home page, which is located here: http://jakarta.apache.org/log4j/.

The org.apache.log4j.Category class

The central component in the log4j 1.1.3 API is the org.apache.logj4.Category class. A category is a named entity and its name is a case-sensitive, hierarchical construct whose naming hierarchy adheres to the following rule, which is taken from the log4j manual:

> A category is said to be an ancestor of another category if its name followed by a dot is a prefix of the descendant category name. A category is said to be a parent of a child category if there are no ancestors between itself and the descendant category.

This is the same convention as the Java package namespace. There exists a special root category that simply is, but has no name. It is accessed via a static method of the Category class. The Category class itself contains a large number of methods, but only the factory, logging and priority state methods are of general interest. A summary of the Category class restricted to these methods is summarized in Listing 1-1.

Listing 1-1, A summary of the key methods in the log4j Category class.

```
public class Category
{
  public static Category getRoot()
  public static Category getInstance(Class clazz)
  public static Category getInstance(String name)
```

```
...
  public void debug(Object msg)
  public void debug(Object msg, Throwable t)
  public boolean isDebugEnabled()
  public void info(Object msg)
  public void info(Object msg, Throwable t)
  public boolean isInfoEnabled()
...
  public boolean isEnabledFor(Priority priority)
  public void log(Priority priority, Object msg)
  public void log(Priority priority, Object msg, Throwable t)
}
```

Before going through the methods we need to define the Priority class that shows up in the Category method signatures. The org.apache.log4j.Priority object represents the importance or level of a message. Whenever you log a message it has a Priority associated with it. There are a small number of Priorities defined by default and are know by the names: FATAL, ERROR, WARN, INFO and DEBUG. You can extend the set of known priorities by providing subclasses of the Priority class. The utility of assigning a priority to a message is that it allows one to filter messages based on their priority or importance. Further, you can test to see if a given priority has been enabled for a Category to avoid generating log messages that would have no affect due to the current priority filters. This is important for high frequency debugging messages whose volume can adversely impact the server. Priority objects have both a string name and an integer value. The name is simply a mnemonic label for the priority. The integer value defines a relative order amongst priorities. This allows one to enable or disable all priorities below a given threshold.

> Log4j 1.2 note, the org.apache.log4j.Priority class has been superceded by the org.apache.log4j.Level class to make the log4j API consistent with the java.util.logging package of JDK 1.4+. The Level class extends Priority and is the preferred construct for representing message importance.

The getRoot method is an accessor for the anonymous root of the default category hierarchy. The getInstance method is a factory method which returns the unique Category instance associated with the given name. If the category does not exist it will be created. The version that accepts a Class simply calls getInstance(clazz.getName()).

The debug, isDebugEnabled, info, and isInfoEnabled methods are convenience methods that invoke the corresponding log or isEnabledFor method with the Priority that corresponds the priority associated with the convenience method. For example, debug(Object) simply invokes log(Priority.DEBUG, Object).

The isEnabledFor(Priority) method checks to see if the Category will accept a message of the indicated Priority. The log(Priority, Object) and log(Priority, Object, Throwable) pass the

message onto the appenders associated with the Category provided that the messages pass the current Priority filter.

org.apache.log4j.Logger

In log4j 1.2, the Category class has been superceded by the org.apache.log4j.Logger class to be more consistent with the JDK 1.4 java.util.logging package. The Logger class is an extension of Category that only adds factory methods for obtaining an org.apache.log4jLogger instance. The key methods of the Logger class are shown in Listing 1-2.

Listing 1-2, A summary of the key methods in the log4j 1.2 Logger class.

```
public class Logger extends Category
{
  public static Logger getLogger(Class clazz)
  public static Logger getLogger(Class clazz)
  public static Logger getRootLogger()
}
```

The Category class has been deprecated in log4j 1.2 and its javadoc warns:

> This class has been deprecated and replaced by the Logger subclass. It will be kept around to preserve backward compatibility until mid 2003.

The JBoss org.jboss.log.Logger wrapper

The JBoss server framework actually uses a simple wrapper around the log4j Category. This wrapper adds support for a custom TRACE level priority and removes the unused Category methods. This does not interfere with the log4j Category usage in any way. The Logger class simply provides a collection of explicit log priority convenience methods as well as a factory method as show in Listing 1-3.

Listing 1-3, The JBoss Logger class summary.

```
package org.jboss.logging;

import org.apache.log4j.Category;
import org.apache.log4j.Priority;

public class Logger
{
   private Category log;

   public static Logger getLogger(String name)
```

```
   public static Logger getLogger(Class clazz)

   public Category getCategory()

   public boolean isTraceEnabled()
   public void trace(Object message)
   public void trace(Object message, Throwable t)

   public boolean isDebugEnabled()
   public void debug(Object message)
   public void debug(Object message, Throwable t)

   public boolean isInfoEnabled()
   public void info(Object message)
   public void info(Object message, Throwable t)

   public void warn(Object message)
   public void warn(Object message, Throwable t)

   public void error(Object message)
   public void error(Object message, Throwable t)

   public void fatal(Object message)
   public void fatal(Object message, Throwable t)

   public void log(Priority p, Object message)
   public void log(Priority p, Object message, Throwable t)
}
```

Not only does this provide direct support for the TRACE level priority used internally by the JBoss server for high-frequency messages that should not normally be displayed, it also avoids the problem of introducing a custom Category factory. In previous versions of JBoss, support for the TRACE priority was done using a custom subclass of Category that added the trace support methods. The problem with the custom subclass is that it tended to result in integration problems like ClassCastException errors with custom user services.

You are free to use the JBoss Logger class if you want to take advantage of the TRACE level priority feature. If you are writing custom MBeans or other services that extend from JBoss classes it is likely that you inherit a Logger instance for use. If you are writing applications that should remain independent of the JBoss classes, then use of the JBoss Logger class should be avoided in place of the standard org.apache.log4j.Category or org.apache.log4j.Logger.

The org.apache.log4j.Appender interface

The ability to selectively enable or disable logging requests based on their category is only a request to log a message. The appenders associated with the category that receives the log message handle the actual rendering of the log message. An appender is a logical message

destination. An appender delegates the task of rendering log messages into strings to the layout instance assigned to the appender. There can be multiple appenders attached to a category, which means that a given message can be sent to multiple destinations. All appenders must implement the org.apache.log4j.Appender interface. This interface imposes the notions of layouts as well as filters and error handlers. A number of appenders are bundled with the log4j framework, including appenders for consoles, files, GUI components, remote socket servers, JMS, Windows event loggers, and remote UNIX syslog daemons. Appenders also exist which allow the rendering of messages to occur asynchronously.

The org.apache.log4j.Layout class

The rendering of a log message into a string representation is delegated to instances of the org.apache.log4j.Layout class. A Layout is a formatter that transforms an org.apache.log4j.spi.LoggingEvent object into a string representation. A Layout can also specify the content type of the string as well as header and footer strings.

Configuring log4j using org.apache.log4j.PropertyConfigurator

That is really all you need to know to use the log4j API to perform logging from components. One large detail missing so far is how to configure log4j. This entails setting the category priorities as well as configuration of the appenders associated with categories. The log4j framework provides support for programmatic configuration as well as configuration using XML and Java properties files. We'll discuss the Java properties file configuration method as this is what JBoss uses in its standard configuration.

The Java properties file based configuration of log4j is handled by the org.apache.log4j.PropertyConfigurator class. The PropertyConfigurator class reads the configuration information for category priority thresholds, appender definitions, and category to appender mappings from a Java properties file. The properties file can be changed at runtime to modify the active log4j configuration. We'll learn the basic syntax of the PropertyConfigurator properties file by discussing the standard JBoss log4j.properties file given in Listing 1-4.

Listing 1-4,Tthe standard JBoss-2.4.x log4j.properties configuration file.

```
# A default log4j properties file suitable for JBoss

### Appender Settings ###
### The server.log file appender
log4j.appender.Default=org.apache.log4j.FileAppender
log4j.appender.Default.File=../log/server.log
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
# Use the default JBoss format
log4j.appender.Default.layout.ConversionPattern=[%c{1}] %m%n
# Truncate if it aleady exists.
```

```
log4j.appender.Default.Append=false

### The console appender
log4j.appender.Console=org.jboss.logging.log4j.ConsoleAppender
log4j.appender.Console.Threshold=INFO
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=[%c{1}] %m%n

### Category Settings ###
log4j.rootCategory=DEBUG, Default, Console

# Example of only showing INFO msgs for any categories under
# org.jboss.util
#log4j.category.org.jboss.util=INFO

# An example of enabling the custom TRACE level priority that is
# used by the JBoss internals to diagnose low level details. This
# example turns on TRACE level msgs for the org.jboss.ejb.plugins
# package and its subpackages. This will produce A LOT of logging
# output.
#log4j.category.org.jboss.ejb.plugins=TRACE#org.jboss.logging.TracePriority
```

The first thing to note is that property names in the file are compound names whose components are separated by periods. This is a common pattern used in property files to group properties together. There are really only two classes of properties being defined in Listing 1-4, appenders (prefix = log4j.appender) and categories (prefix = log4j.category, log4j.rootCategory is a special case for the default root category).

The first section of the file (### Appender Settings ###) defines the log4j appender configuration. Property names that begin with the "log4j.appender" prefix specify properties that apply to Appender instances. The first component in the property name after the log4j.appender prefix is the name of the appender. Thus, the first appender configuration is for the appender named "Default". The log4j.appender.Default property defines the type of appender implementation to use. In this case, the org.apache.log4j.FileAppender is specified. The FileAppender implementation represents a file destination. All properties with the log4j.appender.Default prefix define properties on the Default appender instance. The set of properties one can specify for a given appender depend on the appender type. For the FileAppender the name of the log file, the Layout instance to use, and whether existing log files should be appended to are allowed properties. The log4j.appender.Default.layout.ConversionPattern property is setting the ConversionPattern property value for the log4j.appender.Default.layout property of the FileAppender. The type of the Layout instance was specified to be org.apache.log4j.PatternLayout by the log4j.appender.Default.layout property. Refer to the log4j javadocs for the complete syntax of the format string the PatternLayout class supports.

The log4j.appender.Console properties configure a second appender named Console. This appender sends its output to the System.out and System.error streams of the console in

which JBoss is run. One feature common to most appenders, and illustrated by the Console appender configuration, is the ability to filter out log events whose priority is below some threshold. The log4j.appender.Console.Threshold=INFO setting says that only events with priorities greater than or equal to INFO should be handled by an appender. All other messages should simply be ignored.

The second section of the file (###Category Settings ###) defines the appender to category mappings as well as the category priority thresholds. The root category specification of threshold priority and associated appenders is a special case of the log4j.category grouping of properties, which has the following syntax:

```
log4j.rootCategory=[priority] [(, appenderName)*]
```

So, the log4j.rootCategory entry in Listing 11.4 states that the root category priority threshold is set to DEBUG, and its appenders are Default and Console. The general syntax for the category setting is:

```
log4j.category.category_name=[priority] [(, appenderName)*]
```

There are two commented out examples of the general form. The first states that the org.jboss.util category and its subcategories should filter all messages below the INFO priority level. The second, states that the org.jboss.ejb.plugins category should filter all messages below the custom TRACE#org.jboss.logging.TracePriority priority level. Since log4j does not know which class provides the custom priority implementation, the class must be specified using the "#classname" suffix added to the name of the priority.

Configuring log4j using org.apache.log4j.xml.DOMConfigurator

The XML based org.apache.log4j.xml.DOMConfigurator configuration class offers more flexibility and the benefits, and drawbacks, of an XML based configuration. As we'll describe in the Log4jService MBean configuration section, JBoss supports both the properties file and XML version of the log4j configuration files. For reference, the DTD for the configuration documents supported by the DOMConfigurator of Log4j version 1.1.3 is given in Figure 1-1, and the DTD for Log4j version 1.2.3 is given in Figure 1-2.

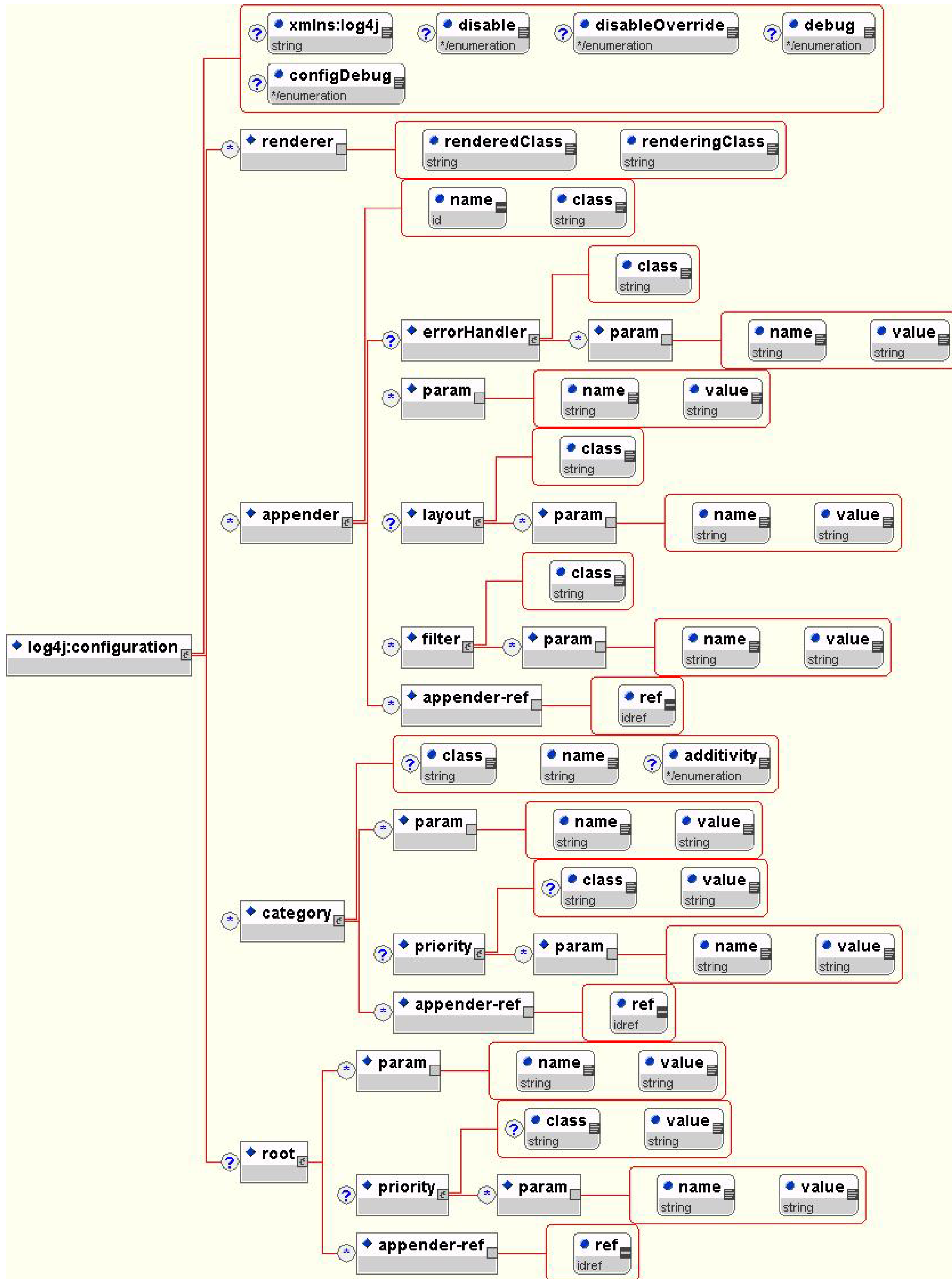
log4j:configuration
xmlns:log4j
string
disable
*/enumeration
disableOverride
*/enumeration
debug
*/enumeration
configDebug
*/enumeration
renderer
renderedClass
string
renderingClass
string
appender
name
id
class
string
errorHandler
class
string
param
name
string
value
string
param
name
string
value
string
layout
class
string
param
name
string
value
string
filter
class
string
param
name
string
value
string
appender-ref
ref
idref
category
class
string
name
string
additivity
*/enumeration
param
name
string
value
string
priority
class
string
value
string
param
name
string
value
string
appender-ref
ref
idref
root
param
name
string
value
string
priority
class
string
value
string
param
name
string
value
string
appender-ref
ref
idref

Figure 1-1, The DTD for the configuration documents supported by the log4j version 1.1.3 DOMConfigurator.

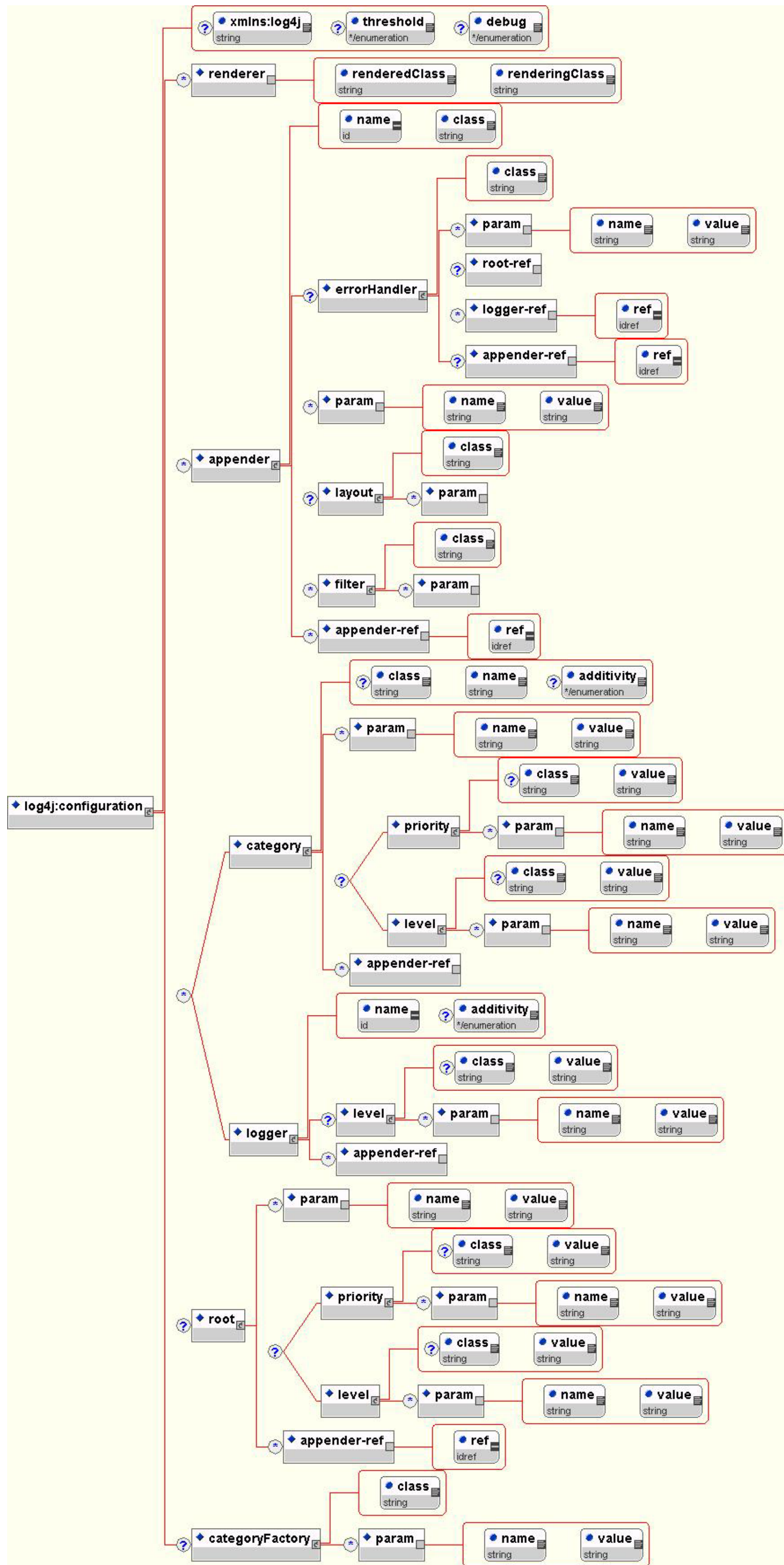
xmlns:log4j
string
threshold
*/enumeration
debug
*/enumeration
renderer
renderedClass
string
renderingClass
string
name
id
class
string
class
string
param
name
string
value
string
root-ref
errorHandler
logger-ref
ref
idref
appender-ref
ref
idref
param
name
string
value
string
appender
class
string
layout
param
class
string
filter
param
appender-ref
ref
idref
class
string
name
string
additivity
*/enumeration
param
name
string
value
string
class
string
value
string
priority
param
name
string
value
string
class
string
value
string
level
param
name
string
value
string
appender-ref
category
log4j:configuration
name
id
additivity
*/enumeration
class
string
value
string
level
param
name
string
value
string
logger
appender-ref
param
name
string
value
string
class
string
value
string
priority
param
name
string
value
string
root
class
string
value
string
level
param
name
string
value
string
appender-ref
ref
idref
class
string
categoryFactory
param
name
string
value
string

Figure 1-2, The DTD for the configuration documents supported by the log4j version 1.2.3 DOMConfigurator.

We'll learn the basic syntax of the DOMConfigurator properties file by discussing the standard JBoss log4j.xml file given in Listing 1-4.

Listing 1-5,Tthe standard JBoss-3.0.x log4j.xml configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<!--
   | For more configuration infromation and examples see the Jakarta Log4j
   | owebsite: http://jakarta.apache.org/log4j
 -->

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">

  <!-- ================================= -->
  <!-- Preserve messages in a local file -->
  <!-- ================================= -->

  <!-- A time/date based rolling appender -->
  <appender name="FILE" class="org.jboss.logging.appender.DailyRollingFileAppender">
    <param name="File" value="${jboss.server.home.dir}/log/server.log"/>
    <param name="Append" value="false"/>
    <!-- Rollover at midnight each day -->
    <param name="DatePattern" value="'.'yyyy-MM-dd"/>
    <!-- Rollover at the top of each hour
    <param name="DatePattern" value="'.'yyyy-MM-dd-HH"/>
    -->

    <layout class="org.apache.log4j.PatternLayout">
      <!-- The default pattern: Date Priority [Category] Message\n -->
      <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
      <!-- The full pattern: Date MS Priority [Category] (Thread:NDC) Message\n
      <param name="ConversionPattern" value="%d %-5r %-5p [%c] (%t:%x) %m%n"/>
       -->
    </layout>
  </appender>

  <!-- A size based file rolling appender
  <appender name="FILE" class="org.jboss.logging.appender.RollingFileAppender">
    <param name="File" value="${jboss.server.home.dir}/log/server.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
  </appender>
```

```
-->

<!-- ============================== -->
<!-- Append messages to the console -->
<!-- ============================== -->

<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
  <param name="Threshold" value="INFO"/>
  <param name="Target" value="System.out"/>

  <layout class="org.apache.log4j.PatternLayout">
    <!-- The default pattern: Date Priority [Category] Message\n -->
    <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c{1}] %m%n"/>
  </layout>
</appender>

<!-- ====================== -->
<!-- More Appender examples -->
<!-- ====================== -->

<!-- Buffer events and log them asynchronously
<appender name="ASYNC" class="org.apache.log4j.AsyncAppender">
  <appender-ref ref="FILE"/>
  <appender-ref ref="CONSOLE"/>
  <appender-ref ref="SMTP"/>
</appender>
-->

<!-- EMail events to an administrator
<appender name="SMTP" class="org.apache.log4j.net.SMTPAppender">
  <param name="Threshold" value="ERROR"/>
  <param name="To" value="admin@myhost.domain.com"/>
  <param name="From" value="nobody@myhost.domain.com"/>
  <param name="Subject" value="JBoss Sever Errors"/>
  <param name="SMTPHost" value="localhost"/>
  <param name="BufferSize" value="10"/>
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="[%d{ABSOLUTE},%c{1}] %m%n"/>
  </layout>
</appender>
-->

<!-- Syslog events
<appender name="SYSLOG" class="org.apache.log4j.net.SyslogAppender">
  <param name="Facility" value="LOCAL7"/>
  <param name="FacilityPrinting" value="true"/>
  <param name="SyslogHost" value="localhost"/>
</appender>
-->

<!-- Log events to JMS (requires a topic to be created)
<appender name="JMS" class="org.apache.log4j.net.JMSAppender">
  <param name="Threshold" value="ERROR"/>
  <param name="TopicConnectionFactoryBindingName" value="java:/ConnectionFactory"/>
```

```
    <param name="TopicBindingName" value="topic/MyErrorsTopic"/>
  </appender>
  -->

  <!-- ================ -->
  <!-- Limit categories -->
  <!-- ================ -->

  <!-- Limit JBoss categories to INFO
  <category name="org.jboss">
    <priority value="INFO"/>
  </category>
  -->

  <!-- Increase the priority threshold for the DefaultDS category
  <category name="DefaultDS">
    <priority value="FATAL"/>
  </category>
  -->

  <!-- Decrease the priority threshold for the org.jboss.varia category
  <category name="org.jboss.varia">
    <priority value="DEBUG"/>
  </category>
  -->

  <!--
     | An example of enabling the custom TRACE level priority that is used
     | by the JBoss internals to diagnose low level details. This example
     | turns on TRACE level msgs for the org.jboss.ejb.plugins package and its
     | subpackages. This will produce A LOT of logging output.
       If you use replace the log4j 1.2 jar with a 1.1.3 jar you will need to
       change this from XLevel to XPriority.
  <category name="org.jboss.system">
    <priority value="TRACE" class="org.jboss.logging.XLevel"/>
  </category>
  <category name="org.jboss.ejb.plugins">
    <priority value="TRACE" class="org.jboss.logging.XLevel"/>
  </category>
  -->

  <!-- ======================= -->
  <!-- Setup the Root category -->
  <!-- ======================= -->

  <root>
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
  </root>

</log4j:configuration>
```

The compound property names of the properties configuration have been replaced by nested XML elements. This makes the complete specification of an item's configuration clearer. Just as in the properties file configuration, in Listing 1-5 there are two types of object being configured, appenders and categories since the log4j:configuration/root element is a special case for the default root category.

The first section of the document is defining the log4j appender configurations. The appender element name attribute defines the name by which the appender may be referenced and the class attribute gives the fully qualified name of the appender implementation. The "FILE" org.jboss.logging.appender.DailyRollingFileAppender is a custom subclass of the org.apache.log4j.DailyRollingFileAppender that simply creates the path to the configured file if it does not exist. This appenders writes log messages to a file and rolls the log over based on its configured date pattern. The properties of the "FILE" appender are specified via child param elements. Each param element gives the name of an appender attribute and the value of the attribute. Note that system property expansion is performed so the value attribute of the param element may reference system property values by enclosing the property name in "${}" as is illustrated by the param element with name='File". The log4j Layout for an appender is specified using a layout child element. The class attribute of the layout element gives the fully qualified class name of the Layout implementation and any attributes of the Layout are specified as nested child param elements.

An example of an Appender configuration that is only possible using the XML based configuration is the appender element named "ASYNC". The org.apache.log4j.AsyncAppender collects log messages and sends them to all the appenders that are associated with it using a separate thread. This is useful with appenders that have high latency like the JavaMail based appender. The appender-ref child elements define the names of the appenders to which the AsyncAppender will send messages.

Configuration of categories is specified using category elements. This allows you to define the appender to category mappings as well as the category priority thresholds. In the standard JBoss configuration there are several examples of setting the priority threshold for different category names. The priority element value attribute gives the name of the priority. To specify a custom priority you also need to include the fully qualified name of the custom Priority implementation, as is done for the "TRACE" priority whose implementation is org.jboss.logging.XLevel.

Establishing the appenders associated with a category is done using the appender-ref child elements. The appender-ref ref attribute value specifies the name attribute value of a previously configured appender element.

Log4j usage patterns

The two biggest usage questions regarding log4j from a developer's perspective are what category names to use, and what message priorities should be used. The pattern used in JBoss is based on the class name of the component performing the logging. In many cases this is the category name used. If there are multiple instances of a component, and it is associated with another meaningful name, this name will be added as a subcategory to the component class name. For example, the org.jboss.security.plugins.JaasSecurityManager class uses a base category name equal to its class name. There can be multiple JaasSecurityManager instances, and each is associated with a security domain name. Therefore, the complete log4j category name used by the JaasSecurityManager is "org.jboss.security.plugins.JaasSecurityManager.securityDomain", where the securityDomain value is the name of the associated security domain.

For message logging priorities, the JBoss usage policy is the following:

- **TRACE**, use the TRACE level priority for log messages that are directly associated with activity that corresponds requests. Further, such messages should not be submitted to a Logger unless the Logger category priority threshold indicates that the message will be rendered. Use the Logger.isTraceEnabled() method to determine if the category priority threshold is enabled. The point of the TRACE priority is to allow for deep probing of the JBoss server behavior when necessary. When the TRACE level priority is enabled, you can expect the number of messages in the JBoss server log to grow at in proportion to N, where N is the number of requests received by the server. The server log may well grow as power of N depending on the request-handling layer being traced.

- **DEBUG**, use the DEBUG level priority for log messages that convey extra information regarding service life-cycle events. Developer or in depth information required for support is the basis for this priority. The important point is that when the DEBUG level priority is enabled, the JBoss server log should not grow proportionally with the number of server requests. Looking at the DEBUG and INFO messages for a given service category should tell you exactly what state the service is in, as well as what server resources it is using: ports, interfaces, log files, etc.

- **INFO**, use the INFO level priority for service life-cycle events and other crucial related information. Looking at the INFO messages for a given service category should tell you exactly what state the service is in.

- **WARN**, use the WARN level priority for events that may indicate a non-critical service error. Resumable errors, or minor breaches in request expectations fall into this category. The distinction between WARN and ERROR may be hard to discern

and so its up to the developer to judge. The simplest criterion is would this failure result in a user support call. If it would use ERROR. If it would not use WARN.

- **ERROR**, use the ERROR level priority for events that indicate a disruption in a request or the ability to service a request. A service should have some capacity to continue to service requests in the presence of ERRORs.

- **FATAL**, use the FATAL level priority for events that indicate a critical failure of a service. If a service issues a FATAL error it is completely unable to service requests of any kind.

This usage policy may indirectly affect your choice of priorities if you log events to the JBoss server appenders. If you do, then you would want to adhere to the above usage policy or you would lose your ability to effectively filter messages in a consistent manner across categories. If you introduce your own appenders for your own category namespace, you are free to choose any priority policy you want as filtering can be done independent from the JBoss categories.

The Log4jService MBean

JBoss provides a org.jboss.logging.Log4jService MBean that manages the configuration of the log4j system. The Log4jService can use either the Java properties style configuration file, or an XML configuration file. The choice between the two is determined solely on the basis of the configuration file name. If the configuration file ends in ".xml", the XML configuration is assumed and the org.apache.log4j.xml.DOMConfigurator class is used. If this is not the case, the configuration file is assumed to be in the Java properties format and the org.apache.log4j.PropertyConfigurator class is used.

In JBoss 2.4.x the Log4jService is loaded as a bootstrap MBean using the standard jboss.conf MLET configuration file, one must specify the service attributes using the MLET constructor syntax. The format used in the default jboss.conf file is:

```
<MLET CODE = "org.jboss.logging.Log4jService"
     ARCHIVE="jboss.jar,log4j.jar"
     CODEBASE="../../lib/ext/">
</MLET>
```

This form uses default values for the log4j configuration file and refresh period. The default configuration file is named "log4j.properties". The default refresh period is 60 seconds. The log4j configuration layer will look to see if the configuration file has changed after each refresh period, and if it has, it will be reloaded. To specify an alternate classpath resource name for the log4j configuration file use:

```
<MLET CODE = "org.jboss.logging.Log4jService"
     ARCHIVE="jboss.jar,log4j.jar"
```

```
     CODEBASE="../../lib/ext/">
   <ARG TYPE="java.lang.String" VALUE="log-config.xml">
</MLET>
```

To specify both a classpath resource name for the log4j configuration file and the refresh period use:

```
<MLET CODE = "org.jboss.logging.Log4jService"
     ARCHIVE="jboss.jar,log4j.jar"
     CODEBASE="../../lib/ext/">
   <ARG TYPE="java.lang.String" VALUE="log-config.xml">
   <ARG TYPE="int" VALUE="180">
</MLET>
```

In JBoss-3.0.x the Log4jService is loaded from the core services configuration file found in the server/<name>/config/jboss-service.xml descriptor.

The configurable attributes of the JBoss-3.0.x Log4jService include:

- **ConfigurationURL**, a URL string specifying location of the
- **RefreshPeriod**, the period in seconds between checks of the ConfigurationURL for changes. The ConfigurationURL is queried for its LastModified value to determine if a change in the configuration has occurred.
- **LoggerPriority**,

```
  <mbean code="org.jboss.logging.Log4jService"
    name="jboss.system:type=Log4jService,service=Logging">
    <attribute name="ConfigurationURL">resource:log4j.xml</attribute>
    <attribute name="RefreshPeriod">60</attribute>
    <attribute name="LoggerPriority"></attribute>
  </mbean>
```

If you need to modify the log4j setup to add your category configuration, you must modify the JBoss server log4j configuration file to add this information.

Summary

This chapter provided an introduction to using log4j with JBoss and the basic configuration syntax for properties file based and XML document based configuration. Next we will cover examples of using log4j to perform monitoring and debugging of the JBoss server.

Chapter

2

2. Using Log4j

Examples of using Log4j with the JBoss server

This chapter will present various Log4j example scenarios to demonstrate the use of Log4j inside of the JBoss server.

Email Notifications Based on Priority

In this section we will look at how to setup JBoss to send notifications via email when logging events occur at or above a specified priority level. This is done using the log4j SMTPAppender.

The SMTPAppender

One very useful Log4j appender is the org.apache.log4j.net.SMTPAppender. This appender allows log messages to be forwarded to an email address through an SMTP gateway. Monitoring for errors and failures is a particularly useful application of this filter. To notify an administrative group you would forward the messages to the group email address. For log messages that correspond to conditions that need immediate attention, the email address used could be an admin cell phone or pager since most cell phone and pager services offer email gateways through which text messages may be sent.

Let's take a look at the SMTPAppender configuration attributes.

- **BufferSize**, set the maximum number of logging events to collect in a cyclic buffer. When the BufferSize is reached, oldest events are deleted as new events are added to the buffer. By default the size of the cyclic buffer is 512 events.

- **EvaluatorClass**, set the fully qualified class name of the class implementing the org.apache.log4j.spi.TriggeringEventEvaluator interface. A corresponding object will be instantiated and assigned as the triggering event evaluator for the SMTPAppender. The appender message buffer is sent whenever the TriggeringEventEvaluator. isTriggeringEvent method returns true. The default evaluator returns true whenever a logging event has a priority >= ERROR.

- **From**, sets the email address to use as the From header for notification messages.
- **SMTPHost**, sets the DNS name or address of the SMTP mail gateway host through which messages will be sent.
- **Subject**, sets the subject line of the email notification messages.
- **To**, the email address to which email notifications are to be sent.
- **Threshold**, sets the minimum priority logging events must have in order to be added to the appender buffer. Note that is independent of the priority used by the evaluator.

An example configuration fragment for the SMTPAppender is given in Listing 2-1.

Listing 2-1, A SMTPAppender configuration sample

```
<!-- Setup the SMTP appender to receive only errors or higher -->
<appender name="ErrorNotifications" class="org.apache.log4j.net.SMTPAppender">

  <param name="Threshold" value="ERROR"/>
  <param name="From" value="nobody-errors@dot.com"/>
  <param name="SMTPHost" value="mailhost.dot.com"/>
  <param name="Subject" value="Error notification"/>
  <param name="To" value="yourname@some.dot.com"/>

  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="[%d{ABSOLUTE},%c{1}] %m%n"/>
  </layout>
</appender>
```

This configuration states that only logging events with priorities of ERROR or higher will be accepted by the appender and that messages will be delivered to "yourname@some.dot.com" with a subject line of "Error notification" through the SMTP gateway host "mailhost.dot.com" and will be identified as being from "nobody-errors@dot.com". Since the threshold value matches the priority of the default evaluator class, messages will be sent as they are added to the appender. If the threshold priority were reduced to WARN, any warning messages would be buffered until an ERROR level message was seen. At that point all messages in the buffer would be sent. Each message in the buffer is delivered in a separate email message.

One thing to note is that when a message is added to the SMTPAppender and it is determined that the message should trigger a send of the current buffer, a mail message is generated for each message in the buffer and delivered using the JavaMail API. This is a relatively slow operation that is performed by the thread that issued the logging event. If this blocking behavior is not what you want, you may use the log4j AsyncAppender to perform the delivery in a background thread.

org.apache.log4j.AsyncAppender

When a message is logged against the SMTPAppender, it composes a mail message using JavaMail. Sending a message is a very slow process compared to logging a message to a file. Typically you do not want to incur this overhead on every error message because the threads in which the errors will be severely impacted performance wise. To avoid this you can make use of the log4j org.apache.log4j.AsyncAppender.

The AsyncAppender configuration attributes include:

- **BufferSize**, set the maximum number of logging events to collect in a FIFO buffer. When the BufferSize is reached, the addition of new messages is blocked until the buffer size falls below the maximum. Therefore, this parameter determines the degree of asynchronous operation between the threads generating the logging events and the background thread. By default the size of the buffer is 128 events.

- **Threshold**, sets the minimum priority logging events must have in order to be added to the appender buffer. The default is to allow all logging events.

The appenders to which the AsyncAppender will delegate messages delivered to it must also be specified. Each appender that should render messages is specified using an appender-ref element in the AsyncAppender configuration. As an example, Listing 2-2 specifies an AsyncAppender configuration for the STMPAppender defined in Listing 2-1.

Listing 2-2, An AsyncAppender configuration sample that delegates messages to the ErrorNotifications SMTPAppender

```
<appender name="ASYNCH" class="org.apache.log4j.AsyncAppender">
  <appender-ref ref="ErrorNotifications"/>
</appender>
```

To control whether logging events are rendered synchronously or asynchronously you need to attach the appender to a category. Typically this is done globally for all categories by attaching the appender to the root category. For example:

```
<root>
  <appender-ref ref="ASYNCH"/>
</root>
```

states that the ASYNCH appender will be sent messages issued through the root category or its descendents.

3. Index

A

Appenders
SMTP ... 21

L

Log4j
Adding your configuration ... 20
Appender ... 7
Appenders ... *See* Appenders
Category ... 4
DOMConfigurator ... 10
DOMConfigurator 1.1.3DTD ... 12
DOMConfigurator 1.2.3DTD ... 14
JBoss Logger ... 6
Layout ... 8
Logger ... 6
MBeans ... 19
PropertyConfigurator ... 8
Supported versions ... 3
Usage patterns ... 18
log4j.properties ... 8
log4j.xml ... 14

N

Notifications
Cell phone, pager ... 21
Email ... 21

P

Pagers ... *See* Notifications
Priorities
Logging ... 18

S

server.log ... 8