ConfiguringTheLogs

Cocoon Logging Configuration

This document describes how to configure Cocoon's logging mechanism. The first section describes the basics of logging, introducing the key concept of Cocoon's logging. The next section describes the configuration files involved in Cocoon's logging.

(An overview of Cocoon logging can also be found here UnderstandingCocoonLogging. EricBoisvert)

Basics

Cocoon's logging mechanism introduces following keywords

- Category is a symbolic name of some sort of logging output.
- Target specifies a physical logging output
- LogEvent created by a logging source, and routed via category resolution to a logging target, the logging sink
- Formatter formats a logging event

Following category names are declared in the cocoon.xconf file

- sitemap
- core.xml-parser
- core.store.transient
- core.store.janitor
- core.xslt-processor
- core.xpath-processor
- core.url-factory
- core.source-handler
- core.program-generator
- core.jsp-engine
- core.i18n-bundles
- core.xscript
- core.language.java
- core.language.js
- core.classloader
- core.markup.xsp
- core.xml-serializer
- core.xml-deserializer
- core.monitor
- core.resolver

A log target is a physical logging destination. A log target is wrapped in a LogTargetFactory, and declared in the logkit.xconf file.

Following log target factories are available

- AsyncLogTargetFactory a virtual target, forwarding log events in a separate thread
- CocoonTargetFactory an enhanced FileTarget, specially tailored for Cocoon needs
- DatagramTargetFactory forwards logging events to datagram sink
- FileTarget write logging events to file sink, offers rotation features
- JDBCTargetFactory writes logging events to JDBC sink
- JMSTargetFactory writes logging events to JMS sink
- PriorityFilterTargetFactory a virtual target, filtering log events by its priority
- SMTPTargetFactory forwards logging events as emails
- ServletTargetFactory forwards formatted log events to the Servlets logging mechanism
- SocketTargetFactory forwards logging events to socket sink
- StreamTargetFactory writes logging events to a java.io.OutputStream, especially to System.out, and System.err

Configuration

Logging categories are used in the java sources. A java programmer decides to use a specific category in some piece of java code.

Alternatively Avalon components used in Cocoon specifies in the its declaration cocoon.xconf the category by the attribute logger.

Note that in both cases we deal only with logging categories, not with logging targets. Logging targets are completely independent from the logging category, in respect of the java programmer.

In next level logkit.xconf maps logging categories to a log target. The configuration file logkit.xconf defines the log targets. More specific, logkit.xconf declares first LogTargetFactories. Next it configures instances of LogTargetFactory objects. And finally it maps logging categories to LogTargetFactory instances.

You have following mapping options

- Map a single category to a single logging target
- Map more than one category to a single logging target
- Map a child category to different logging target

- Define a virtual logging target filtering logging events by priority, and map it to the logging target

Each logging target has some target specific configuration options. Most of the logging targets offer a logging formatting configuration, which are described below.

The configuration file `WEB-INF/web.xml` is only used during startup and until the `logkit.xconf` takes over.

Declaring LogTargets

Before you can configure a specific log target you have to declare it. This is done in the logkit.xconf file.

Inside of the `factories` element each `factory` element declares a log target.

A factory element requires following attributes:

- type - defining the name of the element used for defining this log target instance
- class - specifies the full qualified class name of the LogTargetFactory

Example

This example declares the SerlvetTargetFactory, you may use `servlet` for defining a servlet log target in the following sections of the logkit.xconf file.

```
....
<factories>
  <factory type="servlet"
    class="org.apache.avalon.excalibur.logger.factory.ServletTargetFactory"
  />
...
</factories>
<targets>
...
  <servlet...
  ....
  </servlet>
...
</targets>
....
```

AsyncLogTargetFactory

The configuration of a AsyncLogTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.
- queue-size attribute for specifying the size of queue of log events.
- priority the thread priority of the AsyncLogTarget thread, you may specify MIN, MAX, NORM, or a thread priority integer number, by default the priority is set to 1.

The AsyncLogTargetFactory expects an valid target factory configuration as its child element.

Example

The following configuration template configures a AsyncLogTarget

```
<async-target id="async-target-id" queue-size="15" priority="MIN">
  ... any-target-definition ...
</async-target>
```

CocoonTargetFactory

The configuration of a CocoonTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a CocoonTargetFactory accepts following sub elements:

- filename

- format
- append
- rotation

Example

The following configuration template configures a CocoonTarget, writing log events using the Cocoon's context root directory, as its base directory. (If Cocoon cannot access its context root directory Cocoon sets context-root to the sub-directory log of its work-dir.)

```
<cocoon id="core">
  <filename>${context-root}/WEB-INF/logs/core.log</filename>
  <format type="cocoon">
    %7.7{priority} %{time}   [%{category}] (%{uri})
    %{thread}/%{class:short}: %{message}\n%{throwable}
  </format>
  <append>true</append>
</cocoon>
```

- Note if you change logKit.xconf, you must restart Tomcat*

DatagramTargetFactory

The configuration of a DatagramTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a DatagramTargetFactory accepts following sub elements:

FileTarget

The configuration of a CocoonTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a CocoonTargetFactory accepts following sub elements:

- filename
- format
- append
- rotation

Example

The following configuration template configures a FileTarget, writing logs to the file `/var/tmp/log1.log`.

```
<file>
  <filename>/var/tmp/log1.log</filename>
  <append>false</append>
  <format type="pattern">
    %7.7{priority} %{time} [%{category}] : %{message}\n%{throwable}
  </format>
</file>
```

JDBCTargetFactory

The configuration of a JDBCTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a JDBCTargetFactory accepts following sub elements:

- datasource specify the JNDI data source name
- normalized specify true, or false.
- table specify the name of the table using the attribute name

Setting normalized to true will use the NormalizedJDBCTarget, otherwise the DefaultJDBCTarget is used.

The DefaultJDBCTarget writes each log event into the table specified by the attribute name of element table.

The NormalizedJDBCTarget writes normalized value for category, and priority.

Subelements of table

- category specify the column name as its value,
- static specify the column name as its value, use the attribtue aux to specify the static value
- context specify the column name as its value, use the attribute aux to specify the context entry name.
- message specify the column name as its value
- priority specify the column name as its value
- time specify the column name as its value
- rtime specify the column name as its value
- throwable specify the column name as its value
- hostname specify the column name as its value

Example

The following configuration template configures a JDBC target, writing to table cocoonlog. It uses the JDBC datasource jdbc/logdb. The category value of the log event is written to the column cat of table cocoonlog. The message value of the log event is written to the column msg of table cocoonlog. The priority value of the log event is written to the column prio of table cocoonlog. The time value of the log event is written to the column when of table cocoonlog. The context entry uri is writting to column uri.

```
<database>
  <datasource>jdbc/logdb</datasource>
  <normalized>false</normalized>
  <table name="cocoonlog">
    <category>cat</category>
    <message>mesg</message>
    <priority>prior</priority>
    <time>when</time>
    <context aux="uri">uri</context>
  </table>
</database>
```

JMSTargetFactory

The configuration of a JMSTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a JMSTargetFactory accepts following sub elements:

PriorityFilterTargetFactory

The configuration of a CocoonTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring this target.
- log-level attribute specifies the filter priority level. Logevents of this level or higher are passed further to the enclosed target factory configuration. The log-level sorted from lower priority to highest priority:
    - DEBUG priority defintion 5
    - INFO priority defintion 10
    - WARN priority defintion 15
    - ERROR priority defintion 20
    - FATAL ERROR priority defintion 25

The PriorityFilterTargetFactory expects an valid target factory configuration as its child element.

Example

The following configuration template configures a PriorityLevelTarget

```
<priority-target id="priority-target-id" log-level="ERROR">
  ... any-target-definition ...
</priority-target>
```

SMTPTargetFactory

The configuration of a SMTPTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.
- context-key attribute specifies the context key, storing the required JavaMail Session object, by default the context-key session-context is used.

The configuration of a SMTPTargetFactory accepts following sub elements:

- to one or more recipients addresses
- from the sender address
- subject the subject value
- maximum-size defines the maximum numbers of logevents per mail
- format

Note that the SMTPTargetFactory requires a JavaMail Session object in the context, however, there is no way to place this object in the context when Cocoon starts up. To get around this, add a `<session/>` element into the target configuration. This causes a JavaMail Session object to be created and configured with the child elements of this element, although just an empty element worked fine for me. Here is a sample of an smtp log target:

```
<smtp id="core" context-key="session-context">
  <smtphost>smtp.mail.com</smtphost>
  <to>errors@apache.org</to>
  <from>cocoon@yourdomain.com</from>
  <subject>A log message</subject>
  <maximum-size>10</maximum-size>
  <session/>
</smtp>
```

ServletTargetFactory

The configuration of a SocketTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.
- context-key attribute specifies the context key, storing the required

ServletContext object, by default the context-key attribute uses

the context key servlet-context.

The configuration of a SocketTargetFactory accepts following sub elements:

- format

Example

The following configuration template configures a ServletTarget, expecting a ServletContext object available under context-key servlet-context.

```
<servlet id="servlet-target-id" context-key="servlet-context">
  <format type="pattern">
    %7.7{priority} %{time} [%{category}] : %{message}\n%{throwable}
  </format>
</servlet>
```

SocketTargetFactory

The configuration of a SocketTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a SocketTargetFactory accepts following sub elements:

- address - the attribute hostname specifies the destination host address, and the port attribute spefies the detination port number.

StreamTargetFactory

The configuration of a StreamTargetFactory accepts following attributes:

- id attribute specifies the target name, used in the category definition for referring to this target.

The configuration of a StreamTargetFactory accepts following sub elements:

- stream specifies the stream, you may use following stream sink names
  - System.out outputs log events to the System out stream
  - System.err outputs log events to the System err stream
- format

Example

The following configuration template configures a StreamTarget

```
<stream-target id="stream-target-id">
  <stream>System.out</stream>
  <format type="pattern">
    %7.7{priority} %{time} [%{category}] : %{message}\n%{throwable}
  </format>
</stream-target>
```

L!F5TargetFactory

The L!F5TargetFactory makes use of LogFactor5 - a Swing based GUI that leverages the power of log4j logging framework and provides developers with a sophisticated, feature-rich, logging interface for managing log messages.

More information can be found at the log4j docs.

See also "quick start" LogFactor5 instructions in LogFactorFiveHowto.

Use the following example configuration in order to use the GUI:

```
<logkit>
  <factories>
    ...
    <factory class="org.apache.avalon.excalibur.logger.factory.LF5TargetFactory" type="lf5"/>
    ...
  </factories>
  <targets>
    ...
    <lf5 id="core"/>
  ...
  </targets>
  ...
</logkit>
```

Some notes:

- the GUI starts automatically after the first message is sent to this target
- be aware that the default loglevel at logkit.xconf is ERROR

Formatting

The following section describes the various formatting options available.

PatternFormatter

This formater formats the LogEvents according to a input pattern string.

The format of each pattern element can be

```
%[+|-][#[.#]]{field:subformat}
```

- The +|- indicates left or right justify.
- The #.# indicates the minimum and maximum size of output. You may omit the values and the field will be formatted without size restriction. You may specify #, or #. to only define the minimum size. You may specify .# to only define the maximum size.

- field indicates which field is to be output and must be one of properties of LogEvent. The following fields are currently supported:
  - category Category value of the logging event.
  - context Context value of the logging event.
  - message Message value of the logging event.
  - time Time value of the logging event.
  - rtime Relative time value of the logging event.
  - throwable Throwable value of the logging event.
  - priority Priority value of the logging event.
  - thread Name of the thread which logged the event.
- subformat indicates a particular subformat to use on the specified field, and is currently only supported by:
  - context Specifies the context map parameter name.
  - time Specifies the pattern to be pass to SimpleDateFormat to format the time.

A simple example of a typical PatternFormatter format would be:

```
%{time} %5.5{priority}[%-10.10{category}]: %{message}
```

This would produce a line like:

```
1000928827905 DEBUG [     junit]: Sample message
```

The format string specifies that the logger should first print the time value of the log event without size restriction, then the priority of the log event with a minimum and maximum size of 5, then the category of the log event right justified with a minimum and maximum size of 10, followed by the message of the log event without any size restriction.

ExtendedPatternFormatter

Formatter especially designed for debugging applications. This formatter extends the standard PatternFormatter to add two new possible expansions. These expansions are %{method} and %{thread}. In both cases the context map is first checked for values with specified key. This is to facilitate passing information about caller/thread when threads change (as in AsyncLogTarget). They then attempt to determine appropriate information dynamically.

CocoonLogFormatter

An extended pattern formatter. New patterns are defined by this class are :

- class outputs the name of the class that has logged the message. The optional short subformat removes the package name. Warning : this pattern works only if formatting occurs in the same thread as the call to Logger, i.e. it won't work with

AsyncLogTarget.

- thread outputs the name of the current thread (first element on the context stack).
- uri outputs the request URI.

Logging Rotation

The logging rotation feature is restricted to FileTargets, and logging targets derived from a FileTarget.

The rotation concept introduce a file strategy for defining the filename, orthogonally to the file strategy, a rotate strategy decides the condition of rotation.

Following file strategies are available

- RevolvingFileStrategy write logging data to a fixed number of files
  - attribute type=revolving
  - attribute init specifies starting revolving index, by default it is 0
  - attribute max specifies ending revolving index, by default it is 10
- UniqueFileStrategy, write logging data to unique named file
  - attribute type=unique
  - attribute pattern is SimpleDatePattern, by default it is `yyyy-MM-dd`
  - attribute suffix is optional, by default it is `.log`

Following rotate strategies are available

- RotateStrategyByDate - rotate if the current date, is different from the starting date of logging, or the latest rotation, comparing is performed via formatting both date using the specifed SimpleDateFormat pattern
- RotateStrategyByTime - rotate after logging a period of time, specifed as SimpleDateFormat value of the format HH:mm:SS
- RotateStrategyBySize - rotate if size of logging data exceeds the specified size
- OrRotateStrategy - combining a rotate strategy, rotate if one of the enclosed rotate strategy wants to trigger a rotation

Specifying the rotation settings is done in the target section of a log target.

Specify a revolving file strategy using 4 files, rotate if writing more than 10 MB logging date, or an hour of day in year has expired. The files have suffix like `000001`

```
...
  <rotation type="revolving" init="1" max="4">
    <or>
      <size>10m</size>
      <date>DD:HH</date>
    </or>
  </rotation>
...
```

Specifying a unique file strategy, and rotate if more than 100 MB logging data has been written, or after 5 minutes of logging. The files have suffix like `2002 11261127.log`.

```
<rotation type="unique" pattern="yyyyMMddHHmm" suffix=".log">
  <or>
    <size>100m</size>
    <time>00:05:00</time>
  </or>
</rotation>
```

A short date pattern summary

- yyyy Year
- MM Month, MMM Month by Name
- DD Day in year
- dd Day in month
- HH Hour in day 0-23
- kk Hour in day 1-24
- mm Minute
- ss Second in minute
- SSS Milliseconds

Specifying logkit.xconf

Defining the logkit.xconf file you want to use depends on the Cocoon environment.

- Servlet environment, you specify a servlet init parameter logkit config, eg.

```
...
<init-param>
<param-name>logkit-config</param-name>
<param-value>/WEB-INF/logkit.xconf</param-value>
</init-param>
...
```

- Commandline environment, specify the option `-k`, or `--logKitconfig`, eg.

```
java org.apache.Cocoon -k logkit.xconf ...
```

, or

```
java org.apache.Cocoon --logKitconfig logkit.xconf ...
```

Examples

This snippet configures a file rotation, rotating the file every 5 minutes, or if the file size exceeds 2MB. The filename suffixes represents the date and time of file creation, eg. `access.log20021209 151231.log`.

```
<logkit>
  <factories>
    <factory type="file"
      class="org.apache.avalon.excalibur.logger.factory.FileTargetFactory"
    />
  </factories>
  <targets>
    <file>
      <filename>/tmp/access.log</filename>
      <format pattern="extended">
        %7.7{priority} %{time}   [%{category}]: %{message}\n%{throwable}
      </format>
      <append>false</append>
      <rotation type="unique" pattern="yyyyMMdd HHmmss" suffix=".log">
        <or>
          <size>2m</size>
          <time>00:05:00</time>
        </or>
      </rotation>
    </file>
  </targets>
  <categories>
    <category name="my-cat-1" log-level="DEBUG">
      <log-target id-ref="file"/>
    </category>
  </categories>
</logkit>
```

See Also

- ExploringTheLogs
- JCSLogging
- LogFactorFiveHowto
- AvalonLogKit
- ExcaliburLogger