UnderstandingCocoonLogging

Understanding Cocoon Logging

logkit structure

The logkit.conf file is made of 3 sections. The first section, *factories*, defines a series of Logger class factories (factories are special classes that manage the creation of a logger type) .

For example:

```
<factory class="org.apache.cocoon.util.log.CocoonTargetFactory"
 type="cocoon"/>
```

This logger type is a special kind of logger made for cocoon that log events into a file. Other kind of loggers are possible.

What these logger types actually do are described here in ConfiguringTheLogs

The second section, *targets*, are instances of these logger types. For example, the

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

is a specific log destination of type 'cocoon' (see the previous example factory) named 'sitemap'.. many instance of this kind of logger are possible (for example, a certain factory, like SMTPTargetFactory might create a logger class that sends emails to someone). If you scan the targets section of the logkit.conf, you'll find the familiar list of log files (errors, core,s itemap,handled-errors,etc..). In the default implementation of Cocoon, they are all of cocoon type. The structure of the tags depends of the logger type and there are documented on the ConfiguringTheLogs page.

The last section, "categories", are really like matchers in the sitemap. This sections tells what log event to catch and where the send them (in which target). Categories are recursive, so categories can be made of other categories:

```
<category log-level="WARN" name="core">
      <!-- Startup component manager logger -->
      <category log-level="INFO" name="startup">
        <log-target id-ref="core"/>
        <log-target id-ref="error"/>
      </category>
        ...
      <log-target id-ref="core"/>
      <log-target id-ref="error"/>
</category>
```

This kind of works like a matcher for log events coming from components. For instance, <category log-level="WARN" name="core"> will match all log event sent to core.*, and the embedded category `<category log-level="INFO" name="startup">` will match all log events sent to core.startup.* (since this category is a sub category of 'core'). If, for instance, a log event is sent to core.dummy and there are no such sub category, it will be handle by *core*.

The log-target tags tells where the log event will be logged (the same event can be save in ore that one file, as this example shows). For example, a log event sent to core.startup will be simultaneously logged in core and error targets, therefore in this case, 2 differents files (which are both cocoon loggers types)

The log-levels are filters to adjust the verbosity of the log.. The levels are DEBUG, INFO, WARN, ERROR, FATAL_ERROR in this order. The most verbose is DEBUG because it accepts all the other logs types as well. WARN accepts them all except DEBUG, etc.. and finally FATAL_ERROR only accepts FATAL_ERROR logs.

Configuring component

How does the component knows where to send the log event ??.. well, if you look at any sitemap, you might find declarations of components that looks like this:

```
<map:components>
...
<map:actions>
        <map:action logger="sitemap.action.xsp-action"
        name="xsp-action"
        src="org.apache.cocoon.acting.ServerPagesAction"/>
</map:actions>
...
</map:components>
```

The logger attribute tells that this action (xsp-action) will send the log event to 'sitemap.action.xsp-action'. Looking at the logkit.conf, you should find a sitemap category, and under this category another one called *handled-errors*..

```
<category log-level="INFO" name="sitemap">
      <log-target id-ref="sitemap"/>
      <log-target id-ref="error"/>

      <category log-level="INFO" name="handled-errors">
        <!-- Exceptions that will be handled by a sitemap errorhandler are
             logged to this target. -->
        <log-target id-ref="handled-errors"/>
      </category>
    </category>
```

but no action.xsp-axtion, therefore the log events are handled by the 'sitemap' category, which send the log events to 'sitemap' and 'error' targets (which are both cocoon loggers).

How does it look in the java code ?

The java code is quite simple.. For all Avalon components, you can use the

```
getLogger()
```

method to access the logger and then call a series of methods to send a log event, for example.

```
getLogger().debug("hi !);
```

will send a DEBUG level log to the category define in the logger=".." attribute of the component.

`getLogger().warn("careful !!");` will send a WARN level log, etc..

Complete example

Now, A complete example.. how to set a new logger to debug your component.

1) let's use something else that a cocoon type logger (just for fun).

In the logkit.conf file in the factories section, let's add a new line to declare a new factory (included in cocoon distribution) called a `FileTargetFactory` (it creates a simple log file):

```
   <factories>
     <factory

class="org.apache.avalon.excalibur.logger.factory.PriorityFilterTargetFa
ctor
 y" type="priority-filter"/>
     <factory

class="org.apache.avalon.excalibur.logger.factory.ServletTargetFactory"
 type="servlet"/>
     <factory class="org.apache.cocoon.util.log.CocoonTargetFactory"
 type="cocoon"/>
     <factory
 class="org.apache.avalon.excalibur.logger.factory.LF5TargetFactory"
 type="lf5"/>
<!-- added this factory -->
     <factory class="org.apache.avalon.excalibur.logger.factory.FileTargetFactory"
 type="file"/>

   </factories>
```

The factories section should now look like the previous paragraph. The new type is called **file**

2) Now let's create a target of this type in the targets section. The 'file' tag says this is a logger of 'file' type (created by the FileTargetFactory).

```
      <file id="demo">
       <filename>${context-root}/WEB-INF/logs/demo.log</filename>

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

The tags are pretty self-explanatory, what the fields means is described in ConfiguringTheLogs

3) Now, at the root of the categories sections, let's create a matcher for our log events

```
 <category log-level="DEBUG" name="demo">
       <category log-level="DEBUG" name="transformer">
          <log-target id-ref="demo"/>
       </category>
       <log-target id-ref="demo"/>
       <log-target id-ref="error"/>
     </category>
```

the first level will match all log events sent to demo.*, the second level, all logs sent to demo.transformer.*, etc..

4) Now we need to tell the component where to send the log events. In the sitemap,

```
 <map:components>
 ...
 <map:transformers default="xslt">
 <map:transformer name="MyTransformer" logger="demo.transformer"
         src="org.cocoondev.MyShop.MyTransformer"/>
 </map:transformers>
 ...
 </map:components>
```

and FINALLY !!

You can now use the

```
getLogger().debug(...);
```

call in you java code !

⚠️ Remember that you must restart tomcat for changes in the logkit.conf to be applied.