Using and Implementing Matchers and Selectors (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introduction

Matchers and selectors are sitemap components. They are used to determine the flow, the other components involved and their ordering of the request processing. One particular matcher you're probably familiar with, is the WildcardURIMatcher. That is the one that determines the (sub-)pipeline in the welcome example. But there are many more matchers supplied with Apache Cocoon, one matches the requested URI on regular expressions, others match the client's hostname, existence of parameters and so on.

There is also a number of selectors supplied with Cocoon. Selectors test a generally simple boolean expression and allow to select on roughly the same things as matchers would. There is a selector on the client's hostname, on the client's browser etc.

To make things even more complicated, actions have very similar properties. You can nest other sitemap elements in an action and those are included in the processing only if the action completes successfully.

2. So what are the differences?

The basic structure of a selector is that of a case, switch or if-elseif-...-elseif-else statement in programming languages while matchers and actions compare more to a if statement. In addition selectors don't communicate the basis for their decision to the embedded elements, just select the next part(s) of the pipeline. Matchers and actions, however, add a new map to the environment that can be used for the further processing in the sub pipeline.

You've already come across this feature on the example sitemap: The value matched by the WildcardURIMatcher is used to determine the filename docs/samples/xsp/{1}.xsp. Here {1} represents the value that is stored in the environmental map with the key 1. The name of the key is arbitrary and set by the matcher. If you had supplied a more complex pattern, there would be others. For example <map:match pattern="*/*/*/*/report.html"> would result in keys 1, 2, 3, and 4 being defined, corresponding to the *s in the pattern.

BTW you cannot access those maps from your XSP. Use parameters to the generator to explicitly send them. On your XSP you can access them through an object called parameters. Example

```
<map:match pattern="*/*/*/*/report.html"> <map:generate type="serverpages"
src="docs/getPostcodeData.xsp"> <parameter name="postcode" value="{1}{2} {3}{4}"/>
</map:generate> <map:transform src="stylesheets/html/report.xsl"/> <map:serialize/>
</map:match>
```

On your XSP do

```
<xsp:expr>parameters.getParameter("postcode")</xsp:expr>
```

Generally, one could say that selectors are better suited if the decisions has few easily distinguishable cases, the map feature is not needed and the decision occurs later in the pipeline. Their implementation should be lightweight and so is their integration in the compiled sitemap.

Matchers are often the very first element in a pipeline. They direct the processing based on more complex decision process. They are general purpose but more costly than selectors.

Actions should be used to *"do"* something, not to chose between different sub pipelines. That should

be done only, if an error occurred and you cannot continue the regular pipeline. Of course this is a fuzzy criterion and using an action to choose between exactly two sub pipelines is a very common task i.e. for authentication. Also, often actions have no nested elements and the further processing is not affected by the result of the action.

3. Using Matchers

Like every other sitemap component, matchers need to be declared in the sitemap:

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components> ... <map:matchers default="wildcard"> <map:matcher name="wildcard" src="org.apache.cocoon.matching.WildcardURIMatcher"/> ... <map:matcher name="next-page" src="org.apache.cocoon.matching.WildcardRequestParameterMatcher"> <map:parameter name="parameter-name" value="next-state"/> </map:matcher> </map:matchers> ... </map:components> <map:resources/> <map:pipelines/> </map:sitemap>

Matchers are given a short name (e.g, "wildcard") and of course the name of the JAVA class that implements the matcher. In addition, parameters can be defined as well.

One matcher may be defined as default matcher, so whenever a matcher appears in the sitemap without explicit type specification it will be assumed that it is of the default type.

In a pipeline use the matcher like this

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components/> <map:resources/> <map:pipelines> <map:pipeline> <map:match pattern="*"> <map:generate type="serverpages" src="test/{1}.xsp"/> <map:transform src="stylesheets/dynamic-page2html.xsl"/> <map:serialize/> </map:match> </map:pipeline> </map:pipelines> </map:sitemap>

Matchers can be nested:

<map:match type="sessionstate" pattern="edit*"> <!-- here you could insert parameters for the above matcher --> <map:parameter name="attribute-name" value="__sessionstate"/> <map:match type="next-page" pattern="ok*"> <!-- do something here, eg. database updates --> <map:call resource="simple-page1"/> </map:match> <map:match type="next-page" pattern="delete*"> <!-- do something different here, eg. database deletions --> <map:call resource="simple-page1"/> </map:match> </map:match>

4. Using Selectors

As said above, selectors are very similar to matchers. Again, you need to declare selectors in the sitemap.xmap

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components> ... <map:selectors default="browser"> <map:selector name="browser" src="org.apache.cocoon.selection.BrowserSelector"> <browser name="explorer" useragent="MSIE"/> <browser name="lynx" useragent="Lynx"/> <browser name="netscape" useragent="Mozilla"/> </map:selector> <map:selector name="parameter" src="org.apache.cocoon.selection.ParameterSelector"/> </map:selectors> ... </map:components> <map:resources/> <map:pipelines/> </map:sitemap>

Their use is a bit different to matchers, though:

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components/> <map:resources/> <map:pipelines> <map:pipeline> <map:match pattern="*"> <map:generate

type="serverpages" src="test/{1}.xsp"/> <map:select type="browser"> <!-- you could insert parameters here as well --> <map:when test="explorer"> <map:transform src="stylesheets/w3c-2-msie.xsl"/> </map:when> <map:when test="lynx"> <map:transform src="stylesheets/dynamic-page2html-text.xsl"/> <map:serialize/> </map:when> <map:when test="netscape"> <map:transform src="stylesheets/ns4.xsl"/> </map:when> <map:otherwise> <map:transform src="stylesheets/w3c.xsl"/> </map:otherwise> </map:select> <map:transform src="stylesheets/dynamic-page2html.xsl"/> <map:serialize/> </map:match> </map:pipeline> </map:pipelines> </map:sitemap>

Obviously, this could have been done with matchers as well. Decide on yourself, what appears clearer to you in a specific situation.

5. Write Your Own Matchers and Selectors

5.1. Matchers

Since the basic structure and the assumptions are very similar, we look first at matchers and point out the differences to selectors at the end.

Matchers need to implement the org.apache.cocoon.matching.Matcher interface. See javadocs for more details, see also example matchers included in the distribution.

If you would like to do global configuration for your matcher, it has to implement the org.apache.avalon.framework.configuration.Configurable interface.

Local configuration parameters are avalon parameters and thus can be easily read and used with the generated matcher method.

If the matcher returns not null, the match was successful and the nested sub pipeline is executed. Components in sub pipeline can access the matching result through the returned map.

The easiest way to write your own matcher would be to base it upon org.apache.cocoon.matching.WildcardURIMatcher and override the getMatchString method with your own.

5.2. Selectors

Selectors and selector factories differ from their matcher counter parts only in the fact that selectors return boolean values rather than maps. Thus when a selector returns true the nested elements will be included in the processing, otherwise they are not included. Since no map is returned, no additional information may be passed to those elements.

For selectors, the last argument reads param instead of parameters.

1. Comments

add your comments