Views (2.1 legacy document)

Table of contents

1 Comments.....6

Table of contents

1 The Views....3
1.1 Motivation....3
1.2 Defining A View....3
1.3 Placing Labels....4
1.4 Placing Labels Summary....5
1.5 View Processing....5
1.6 Exit Points....5
1.7 Requesting A View....6

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. The Views

Apache Cocoon provides "views" to a resource. Defining a pipeline in a sitemap specifies the different stages of processing a resource. A view defines an exit point in the pipeline processing.

Views are yet another sitemap component. Unlike the rest, they are not inherited by sub-sitemaps and they are orthogonal to the resource and pipeline definitions. In the following we will not distinguish between resources and pipelines because their differences are not relevant here. So, when we talk about pipelines the said is valid for resources as well.

Basically, views let you specify exit points of your pipelines that are taken whenever a particular view is requested. The processing continues with the definitions in the requested view. The advantage over selectors that could achieve the same is, that these exit points are not necessarily declared for each pipeline individually, but once per sitemap.

1.1. Motivation

Implementing a semantic search feature is the main motivation to offer views in Apache Cocoon.

A sitemap defines the URI space of a Cocoon application. Apache Cocoon offers a fairly sophisticated URI space mapping mechanism. Defining pipelines in a sitemap you define this mapping. It's generally a mistake to map a file system (or a directory server, or a database repository) one-to-one with the URI space, since it leads to easily broken links and potential security issues.

Browsers requests resources of this URI space. The response of a browser request is normally intended for presenting to a human being. It may be augmented with navigation controls, and advertisements. An indexer of a search engines requests resources of this URI space, too. In contrast to a browser an indexer is interested in the bare content of a resource.

Views can access the original content of a resource. For example, you can now index a document resource, requesting the "content" view of the resource lacking the aggregation with navigation controls, and advertisements. You can now index the text inside a logo, if you are given the SVG content that generated the raster image. You can index the PDF content without having to implement a PDF parser since you request the "content" view of the resource obtaining an easily parsable XML file.

1.2. Defining A View

You declare a view in the sitemap. The definition of a view may define further processing steps. You are not allowed to define a generator step for a view, as the content of a view is the xml content from a view's exit point. The most simple view just serializes the xml content to xml.

A view element is identified by its name, and its label. You specify the name of a view by the attribute name, and its label either by the attribute from-label, or by the attribute from-position. The following list explains the attributes in more detail.

- The attribute name specifies the name of view. Each view must have a unique name. If you request a view you have to specify the view name.
- The attribute from-label defines a label name of a pipeline exit point. You can choose any name you like, as a label name, except "first", and "last".
- The attribute from-position may have only following values "first", and "last". The special label

"first" is the pipeline exit point after the generator processing stage. The special label "last" is the pipeline exit point right before the serializer processing stage. The labels "first", and "last" are set by the sitemanager automatically.

The following example is a simple resource view, just serializing the xml content of the view's exit point.

```
<map:views> <map:view name="content" from-label="content"> <map:serialize type="xml"/> </map:view>
```

The next example performs an xslt transformation before serializing to xml.

```
<map:views> <map:view name="dublin-core" from-label="dublin-core"> <map:transform src="stylesheets/document2dublin-core.xsl"/> <map:serialize type="xml"/> </map:view>
```

The last example defines a view specifying the position-from attribute "last", and serializing to pdf, for a pdf-aware only indexer, or archiver.

```
<map:views> <map:view name="full-document-content" from-position="last"> <map:transform src="stylesheets/document2fo.xsl"/> <map:serialize type="fo2pdf"/> </map:view>
```

1.3. Placing Labels

You place labels to define a pipeline exit point. A pipeline exit point may be shared by more than a single view.

Defining a pipeline exit point you have to add an attribute "label" to a sitemap element. The following sitemap elements are label aware:

Sitemap Element	Description
map:generator	A generator element is allowed to have a label, eg. <map:generator name="foo" src="bar" label="content"/>. The xml content produced by the generator is passed to the requested view. Moreover requesting a "first" view has the same effect as labelling the first generator of pipeline.
map:generate	A generate element is allowed to have a label attribute, eg. <map:generate type="foo" label="content"/>. The xml content produced by the generator is passed to the requested view. Moreover requesting a "first" view has the same effect as labelling the first generator of pipeline.
map:transformer	A transformer element is allowed to have a label attribute, eg. <map:transformer name="foo" src="bar" label="augmented-content"/>. The xml content produced by the transformer is passed to the requested view.
map:transform	A transform element is allowed to have a label attribute, eg. <map:transform type="foo" label="augmented-content"/>. The xml content produced by the transformer is passed to the requested view.
map:aggregate	An aggregte element is allowed to have a label attribute, eg. <map:aggregate element="foo" label="all-news"/>. The xml content produced by

	the aggregate is passed to the requested view.
map:part	A part element of an aggregate element is allowed to have a label attribute. eg. <map:part src="foo" label="news-only"/>. The xml content produced by the part only is passed to the requested view.

A label attribute may hold 1 or more label names, separated by comma or blank, eg. <map:generate src="foo" label="label1, label2"/>.

1.4. Placing Labels Summary

As described above you have a wide range of choice for placing labels. You may even place labels to part elements, and to pipelines being the source of a labelled part element. The following paragraphs summarize some of the hot features.

You can use more that one label-value (label="content,link rdf") separated by comma or blank.

The aggregate element can have a label attribute which acts as on a generator, or transformer (all part elements are collected).

Part elements can have a label attribute. In this case only those parts are collected which corresponds to the requested view.

If you refer to sources via the cocoon:/ protocol the requested view will be propagated.

The element label is deprecated, and being not supported anymore. Thus you have to rewrite your sitemap if you are using <map:label name="foobar"/>.

Rewrite your sitemap if you were using label elements, moving the label name up to the previous xml producer. For example rewrite your sitemap:

... <map:generate src="foo"/> <map:transform type="bar"/> <map:label name="mylabel"/> <map:serialize/> ...

...to this sitemap:

... <map:generate src="foo"/> <map:transform type="bar" label="mylabel"/> <map:serialize/> ...

1.5. View Processing

The samples sitemap contains two view definitions. One of them looks like the excerpt below.

<map:views> <map:view name="content" from-label="content"> <map:serialize type="xml"/> </map:view>

It only defines what processing steps should be taken, after some exit point labelled "content" is reached. In all this case just a serializer is used to further process the document.

1.6. Exit Points

A look at the pipelines reveals no label "content". But a closer look at the defined components show this:

<map:components> <map:generators default="file"> <map:generator name="file" src="org.apache.cocoon.generation.FileGenerator" label="content"/> <map:generator name="directory" src="org.apache.cocoon.generation.DirectoryGenerator" label="content"/>

<map:generator name="serverpages"
src="org.apache.cocoon.generation.ServerPagesGenerator" label="content"/> ...

Here a number of generators are declared, each one has a label attribute. Now, everytime one of these generators is used in a pipeline, an exit point "content" is generated, just after the generator has been executed.

This is not limited to generators - every sitemap component can be augmented with a view label.

Two special labels exist: "first" and "last". These are automatically declared for every pipeline, after the first component and after the last respectively. This is used by the second view in the samples sitemap.

<map:view name="links" from-position="last"> <map:serialize type="links"/> </map:view>

1.7. Requesting A View

The current way for Cocoon to access views is fixed as a special URI query parameter cocoon-view.

For example querying the view content of the page:

http://localhost:8080/cocoon/documents/index.html

You use following URL:

http://localhost:8080/cocoon/documents/index.html?cocoon-view=content

Suggestions for further accessing views are:

- React on a "variant" HTTP header (nothing cocoon specific since the concept could be impelemented later on by other publishing frameworks).
- React on URI extension: for example http://host/path/file.view, that is something that can be done by configuring the sitemaps manually. (where http://host/path/index is the default resource, and index.content is the XML view of the content).

Since views are orthogonal to pipelines, keep in mind to remove any unwanted view from a deployed application.

1. Comments

add your comments