Understanding Apache Cocoon (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. Overview

This document is intended for both Users and Developers and presents an overall picture of @Name@.

2. Prerequisites

What You Should know:

- XML, XML Namespaces
- Basics of XPath, XSLT
- Java language
- Servlets, HTTP

What You need not know:

- Cocoon 1

3. A Little History

3.1. Cocoon 1

- Cocoon project was founded in Jan. 1999 by Stefano Mazzocchi as an open source project under Apache Software Foundation.
- Started as a simple servlet for XSL styling of XML content.
- Was based on DOM level 1 API. This choice turned out to be quite limiting for speed/memory efficiency.
- Used reactor pattern to connect components. This allowed the reaction instructions to be placed inside the documents. Though appealing, it caused difficulties in managing highly dynamic web-sites.
- Allowed context overlap to happen by having processing instructions in documents/stylesheets.

3.2. Apache Cocoon

- A separate codebase to incorporate Cocoon 1 learnings.
- Designed for execution speed/memory efficiency and scalability to process very large documents by switching processing model from DOM to SAX.
- Centralizes the management functions by allowing processing pipeline specification in a sitemap (an XML file), replacing the embedded processing instruction model.
- Better support for pre-compilation, pre-generation and caching for better performance.

4. What problem does Cocoon solve?

Basic problem to be solved:

4.1. Separation of content, style, logic and management functions in an XML content based web site (and web services).

The @Name@ Pyramid Model of Contracts

4.2. Data Mapping

The @Name@ Data Mapping

5. Basic Mechanisms.

Basic mechanisms for processing XML documents:

- Dispatching based on Matchers.
- Generation of XML documents (from content, logic, Relation DB, objects or any combination) through Generators
- Transformation (to another XML, objects or any combination) of XML documents through Transformers
- Aggregation of XML documents through Aggregators
- Rendering XML through Serializers

Basic Mechanisms

5.1. Pipeline Processing

Sequence of Interactions

Interaction Sequence

Pipeline

Pipeline

6. Architecture.

Architecture

6.1. Core Cocoon

- Avalon framework for logging, configuration, threading, context etc.
- Caching mechanism
- Pipeline handling
- Program generation, compilation, loading and execution.
- Base classes for generation, transformation, serialization, components.
- ...

6.2. Cocoon Components

- Specific generators
- Specific transformers
- Specific matchers
- Specific serializers
- ...

6.3. Built-in Logicsheets

- sitemap.xsl
- xsp.xsl
- esql.xsl
- request.xsl
- response.xsl
- ...

6.4. Site specific configuration, components, logicsheets and content

- ...

7. Abstraction.

7.1. eXtensible Server Pages (XSPs)

An XSP page is an XML page with following requirements:

- The document root must be <xsp:page>
- It must have language declaration as an attribute in the <xsp:page> element.
- It must have namespace declaration for xsp as an attribute in the <xsp:page> element.
- For an XSP to be useful, it must also require at least an <xsp:logic> and an <xsp:expr> element.

<?xml version="1.0" encoding="ISO-8859-1"?> <xsp:page language="java" xmlns:xsp="http://apache.org/xsp"> <xsp:logic> static private int counter = 0; private synchronized int count() { return counter++; } </xsp:logic> <page> I have been requested <xsp:expr>count()</xsp:expr> times. </page> </xsp:page>

An XSP page is used by a generator to generate XML document.

7.2. XSP Processing (Code Generation)

package org.apache.cocoon.www.docs.samples.xsp; import java.io.File; // A bunch of other imports public class counter_xsp extends XSPGenerator { // .. Bookkeeping stuff commented out. /* User Class Declarations */ static private int counter = 0; private synchronized int count() { return counter++; } /* Generate XML data. */ public void generate() throws SAXException { this.contentHandler.startDocument(); AttributesImpl xspAttr = new AttributesImpl(); this.contentHandler.startPrefixMapping("xsp", "http://apache.org/xsp"); this.contentHandler.startElement("", "page", "page", xspAttr); // Statements to build the XML document (Omitted) this.contentHandler.endElement("", "page", "page"); this.contentHandler.endPrefixMapping("xsp"); this.contentHandler.endDocument(); }

7.3. Ways of Creating XSPs

7.3.1. Embedded Logic

- Code is embedded in the XML page
- No separation of content and logic
- Okay for small examples but terrible for large systems.

ways of creating xsp's

7.3.2. Included Logicsheet

- Code is in a separate logicsheet (an XSL file)
- Effective separation of content and logic
- Preferred way to create XSPs

ways of creating xsp's

7.3.3. Logicsheet as tag library

- The logicsheet is packaged as a reusable tag library and registered with Cocoon in cocoon.xconf file.
- Tag library has a namespace declaration, declared in the original logicsheet and matched in <xsp:page> xmlns:... attribute.
- Effective separation of content, logic and management

ways of creating xsp's

7.4. Sitemap

<?xml version="1.0"?> <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components> ... </map:components> <map:views> ... </map:views> <map:pipelines> <map:pipeline> <map:match> ... </map:match> ... </map:pipeline> ... </map:pipelines> ... </map:sitemap>

Sitemap contains configuration information for a Cocoon engine:

- list of matchers
- list of generators
- list of transformers
- list of readers
- list of serializers
- list of selectors
- list of processing pipelines with match patterns
- ...

Sitemap is an XML file corresponding to a sitemap DTD.

Sitemap can be edited to add new elements.

Sitemap is generated into a program and is compiled into an executable unit.

7.5. Matchers

A Matcher attempts to match an URI with a specified pattern for dispatching the request to a specific processing pipeline.

Different types of matchers:

- wildcard matcher
- regexp matcher

More matchers can be added without modifying Cocoon.

Matchers help in specifying a specific pipeline processing for a group of URIs.

Sitemap entries for different types of matchers

<map:matchers default="wildcard"> <map:matcher name="wildcard" factory="org.apache.cocoon.matching.WildcardURIMatcher"/> <map:matcher name="regexp" factory="org.apache.cocoon.matching.RegexpURIMatcher"/> </map:matchers>

Pipeline entries in sitemap file

<map:match pattern="jsp/*"> <map:generate type="jsp" src="/docs/samples/jsp/{1}.jsp"/> ... </map:match> <map:match pattern="hello.pdf"> </map:match

7.6. Generators

A Generator is used to create an XML structure from an input source (file, directory, stream ...)

ways of creating xsp's

Different types of generators:

- file generator
- directory generator
- XSP generator
- JSP generator
- Request generator
- ...

More generators can be added without modifying Cocoon.

Sitemap entries for different types of generators

<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"/> <map:generator name="request" src="org.apache.cocoon.generation.RequestGenerator"/> ... </map:generators>

A sample generator entries in a pipeline

<map:match pattern="hello.html"> <map:generate src="docs/samples/hello-page.xml"/> <map:transform src="stylesheets/page/simple-page2html.xsl"/> <map:serialize type="html"/> </map:match>

A Generator turns an XML document, after applying appropriate transformations, into a compiled program whose output is an XML document.

An XSP generator applies all the logicsheets specified in the source XML file before generating the program.

Generators cache the compiled programs for better runtime efficiency.

7.7. Transformers

A Transformer is used to map an input XML structure into another XML structure.

Different types of transformers:

- XSLT Transformer
- Log Transformer
- SQL Transformer
- I18N Transformer
- ...

Log Transformer is a good debugging tool.

More transformers can be added without modifying Cocoon.

Sitemap entries for different types of transformers

```
<map:transformers default="xslt"> <map:transformer name="xslt"
src="org.apache.cocoon.transformation.TraxTransformer">
<use-request-parameters>false</use-request-parameters>
<use-browser-capabilities-db>false</use-browser-capabilities-db> </map:transformer>
<map:transformer name="log" src="org.apache.cocoon.transformation.LogTransformer"/> ...
</map:transformers>
```

A sample transformer entry in a pipeline

```
<map:match pattern="hello.html"> <map:generate src="docs/samples/hello-page.xml"/>
<map:transform src="stylesheets/page/simple-page2html.xsl"/> <map:serialize type="html"/>
</map:match>
```

7.8. Serializers

A Serializer is used to render an input XML structure into some other format (not necessarily XML)

Different types of serializers:

- HTML Serializer
- FOP Serializer
- Text Serializer
- XML Serializer
- ...

More serializers can be added without modifying Cocoon.

Sitemap entries for different types of serializers

```
<map:serializers default="html"> <map:serializer name="xml" mime-type="text/xml"
src="org.apache.cocoon.serialization.XMLSerializer"/> <map:serializer name="html"
mime-type="text/html" src="org.apache.cocoon.serialization.HTMLSerializer"/>
<map:serializer name="fo2pdf" mime-type="application/pdf"
src="org.apache.cocoon.serialization.FOPSerializer"/> <map:serializer name="vrml"
mime-type="model/vrml" src="org.apache.cocoon.serialization.TextSerializer"/> ...
</map:serializers>
```

A sample serializer entry in a pipeline

```
<map:match pattern="hello.html"> <map:generate src="docs/samples/hello-page.xml"/>
<map:transform src="stylesheets/page/simple-page2html.xsl"/> <map:serialize type="html"/>
</map:match>
```

7.9. Pipeline Processing

The sitemap configuration allows dynamic setup of processing pipelines consisting of a generator, multiple transformers and a serializer.

Requests are dispatched to a pipeline based on request URI and the pipeline matching pattern (either with wildcards or as a regexp)

The pipeline is setup in the generated file sitemap_xmap.java (This file gets generated [possibly asynchronously] everytime the sitemap.xmap is modified.

Pipeline Entry

7.10. Logicsheets

Logicsheets are XSL files with an associated namespace.

Primary mechanism to add program logic (code) to XSPs.

These need to be registered in configuration file cocoon.xconf.

Logicsheets are used by the generator to transform XML structure before generating program.

Cocoon comes with a no. of built-in logic sheets:

- request.xsl
- response.xsl
- session.xsl
- cookie.xsl
- esql.xsl
- log.xsl
- ...

Log.xsl structure

```
<xsl:stylesheet version="1.0" xmlns:xsp="http://apache.org/xsp"
xmlns:log="http://apache.org/xsp/log" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="log:logger"> ... variable and xsp:logic statements ... </xsl:template>
<xsl:template match="log:debug"> <xsp:logic> if(getLogger() != null)
getLogger().debug("<xsl:value-of select="."/>"); </xsp:logic> </xsl:template> <xsl:template
match="log:error"> ... </xsl:template> </xsl:stylesheet>
```

A sample use

```
<xsp:page language="java" xmlns:xsp="http://apache.org/xsp"
xmlns:log="http://apache.org/xsp/log"> <page> <log:logger name="test" filename="test.log"/>
<log:debug>Test Message</log:debug> </page> </xsp:page>
```

8. @Name@ Configuration.

Cocoon is highly configurable. Main configuration files, assuming Cocoon deployment as a servlet in a servlet container, are (directory locations assume Tomcat servlet container):

- sitemap.xmap: the sitemap file. By default, located in $TOMCAT_HOME/webapps/cocoon directory.
- cocoon.xconf: configuration file having logicsheet registrations. Specifies, sitemap.xmap location and other such parameters. By default, located in $TOMCAT_HOME/webapps/cocoon directory.
- web.xml: servlet deployment descriptor. Specifies location of cocoon.xconf, log file location and other such parameters. Located in $TOMCAT_HOME/webapps/cocoon/WEB-INF directory.
- cocoon.roles: mapping file for Core Cocoon components name and implementation classes. For example, if you want to use a parser other than the default one, you need to modify this file.

9. @Name@ Work Area

Cocoon produces execution log entries for debugging/auditing.

- The amount of data to be logged can be controlled by log-level parameter in web.xml file. The default is DEBUG (maximum data).
- By default, the log file is: $TOMCAT_HOME/webapps/cocoon/WEB-INF/logs/cocoon.log.

Cocoon keeps the generated .java files in a directory tree starting at (by default): $TOMCAT_HOME/webapps/work/localhost_8080%2Fcocoon/org/apache/cocoon/www.

You can find sitemap_xmap.java here.

Files created by LogTransformer are kept (by default) in $TOMCAT_HOME directory.

10. Use with Tomcat

Download Tomcat from Apache site.

Download Cocoon sources from Apache CVS. [Command assume UNIX Bourne shell]

export CVSROOT=:pserver:anoncvs@cvs.apache.org:/home/cvspublic cvs login Password: anoncvs cvs checkout cocoon-2.1

Build sources as per instruction in Install file.

Move the cocoon.war file to $TOMCAT_HOME/webapps directory.

Start the servlet engine. Type-in the URL http://localhost:8080/cocoon in your browser. You should see the Cocoon welcome message.

Consult Install file if you face problems.

1. Comments

add your comments