Offline Page Generation (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

Cocoon can generate static, 'offline' versions of web pages or web sites, as well as sites served dynamically. This document covers the concepts involved in offline page and site generation.

2. Offline Page Generation

Cocoon allows static versions of Cocoon web sites to be created.

At present, this can be done in three ways:

- Command Line Interface
- Using Ant Task
- Cocoon Bean

This document explains the general concepts that are shared by all of these approaches. The specific details for each method are explained on a separate page.

Cocoon, when generating pages offline, can follow links in a page (whether that page is HTML, PDF or anything else), and can rewrite URIs to create filenames by checking the mime type of the generated page. All links to pages who's URIs change are changed too.

3. Configuration

To use Cocoon in its offline mode, a servlet container (e.g. Tomcat or Jetty) is not needed. Cocoon can generate an offline site directly using the information available in the Cocoon webapp folder.

Having said this, many choose to have a servlet container available locally for use whilst debugging, as this can speed up the development process significantly.

3.1. Directories and Files

As all the information Cocoon needs to generate a site is stored in the Cocoon webapp directory, we need to tell it where to find it, and where to find various other files and directories. These are:

- Context directory (the Cocoon Webapp directory)
- Configuration File (usually ${COCOON_WEBAPP}/WEB-INF/cocoon.xconf)
- Work Directory (used by Cocoon to store temporary files, this can be anywhere of your choosing)

3.2. Logging

There are three options that need to be specified in relation to logging. These are:

- Log Kit (the logging configuration file, usually ${COCOON_WEBAPP}/WEB-INF/logkit.xconf)
- Logger (a category used for logging, as configured in the configuration file)
- Log Level (a logging level, either DEBUG, INFO WARN, ERROR or FATAL_ERROR. Relates specifically to logging at startup, after which log kit configuration takes over)

3.3. Other Configuration Options

In online mode, a User agent string tells Cocoon what browser is being used to access a page. The user agent can be configured manually for offline generation.

In online mode, an accept string is provided by a browser, telling the browser what types of content it is capable of accepting. This will be a comma separated list of mime types. In offline mode, an accept string can also be specified.

As Cocoon based sites can change the content they generate based upon the user agent string and the accepts string, it can be necessary to specify them in order to have the correct content generated.

In order to generate sites that make use of databases and database connections, it is necessary to load JDBC classes at startup. Cocoon allows for this.

When, in offline mode, Cocoon generates a page ending in a /, the resultant file cannot be written to a filesystem as its name would refer specifically to a directory. Therefore, the user can specify a default filename which will be appended to the page's URI before saving to disc.

4. URIs and Targets

4.1. SourceURIs

A source URI (which may also have a source prefix prepended) is the part of the URI that is given to Cocoon for processing. So, for example, if you access a page with: http://localhost:8080/cocoon/site/page.html then the source URI would be site/page.html

4.2. Destinations and Modifiable Sources

Most of the time, when generating pages, the generated pages will be simply written to disk.

However, this is not the only option. Generated pages can be written anywhere for which a ModifiableSource exists. So, for example, it is possible to generate a site and have the pages written directly to a web server using FTP, by making use of the Avalon FTPSource.

4.3. Target Types

When generating a page, Cocoon needs to know how to decide upon the URI of the generated page. This process could be described as 'URI arithmetic'.

Source and destination URIs are made up of the following elements:

- Source Prefix: Part of a source URI used to request a page but excluded from the destination URI
- Source URI: Part of a source URI that is used when calculating the destination URI
- Destination URI: The base URI for a destination
- Type: The method used for merging the above elements (can be append, replace or insert

When combining elements to make a URI, it is the user's responsibility to include directory separators. For example, foo with bar appended will be foobar, whereas foo/ with bar appended will be foo/bar.

4.3.1. Appending

Here, when calculating the destination URI, the source prefix is ignored, and the destination URI is calculated by appending the source URI to the end of the destination URI. For example, with the following values:

Source prefix: site/, source URI: page.html, destination URI: pages/

A request will be made to Cocoon for a page at: site/page.html. This will be saved as pages/page.html.

4.3.2. Replacing

Here, when calculating the destination URI, the source prefix and the source URI are ignored, and the destination URI is used as is. This is useful when you wish to save the generated page with a filename that bears no relationship to the source URI. For example, with the following values:

Source prefix: site/, source URI: page.html, destination URI: pages/simple.html

A request will be made to Cocoon for a page at: site/page.html. This will be saved as pages/simple.html.

Given the nature of this target type, it inherently cannot be used when following links (otherwise all pages will be written on top of each other).

4.3.3. Inserting

Here, when calculating the destination URI, the source prefix is ignored, and the source URI is inserted into the destination URI at the point marked by an asterisk (*). This is intended for use with complex protocols where the source URI does not appear at the end of the destination URI.

4.4. Mime Type Checking

Cocoon can optionally test the mime type for a page, and, if the mime type doesn't match the page's extension, amend the destination URI to include the correct extension. This will ensure that pages will load correctly when served by a static web server.

When Cocoon amends a destination URI, it also amends URIs for links in those pages, so that links will still work when a site has been crawled.

This feature substantially slows down page generation, as each page must be generated three times, (once to find links, once to find its mime-type and once to collect the actual content. This can be avoided by ensuring that all URIs in the site are correct and do not need amending, in which case it is only necessary to generate a page once.

5. Following Links and Site Crawling

Cocoon can be configured to either follow, or ignore, links in pages that it generates. It has two methods of gathering links, 'link view' and 'link gathering'.

5.1. Link View Crawling

With link view crawling, Cocoon gets the links by generating the 'link view' for a page. Using link view gives a significant degree of configurability in terms of which links are gathered, as it is possible to insert a transformer into the view to select out links that should not be followed.

The disadvantage with link view crawling is that each page must be generated twice, which doubles page generation time.

Link view is usually configured in the root sitemap with:

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

If you have this in your root sitemap, you do not need it in your sub-sitemaps. However, you may choose to override it with one that carries our further processing - for example, with an XSLT transformer that removes links that should not be crawled.

See views for more on views.

You can see the link view yourself by appending ?cocoon-view=links to the page's URI.

5.2. Link Gathering Crawling

With link gathering crawling, links are gathered from the SAX stream right before the serializer. All src, href and xlink:href attributes are taken to be links, and are therefore followed.

The benefit of link gathering crawling is that pages do not need to be generated twice. However, one looses the ability to configure which links should be followed that exists with link view crawling.

6. Broken Links

When a page cannot be found at a URI that has either been specified, or has been found as a link in another page, it is considered 'broken'.

Exactly what is done when a broken link is found depends upon the method used to evoke Cocoon. See related pages for specific details.

6.1. Broken Link Handling using xconf Configuration method

The xconf method allows for more sophisticated broken link handling. The user can select to have broken links reported to a file, this file being either text or XML.

When this file is plain text, it will have one link URI per line.

When this file is in XML, it will detail a message explaining the reason for the broken link, as well as the URI of the link.

It is also possible to specify whether an error page should be generated in the place of the broken page (based upon the configured <map:handle-errors> code in the sitemap). If required, an extension can be appended to the original file's URI to signify that it is an error page (e.g. .error).

7. Precompiling XSPs

When used offline, Cocoon can precompile XSP pages. If no URIs are specified, it will scan all directories within the context directory looking for XSP files, each of which will be compiled. If URIs are specified, all links will be followed looking for pages that make use of XSP, compiling those XSP pages as they are found.

1. Comments

add your comments