

CssColumnSite Application

Bernhard Huber, berni_huber@a1.net

1 CssColumnSite Application

This document describes as Cocoon setup for a simple static publishing site.

2 Requirements

A simple but flexible site should be built using Cocoon. Following requirements triggered the design:

- Use Cocoon
- Use HTML layout
- Try to avoid using HTML tables for layouting
- Generate static HTML for uploading to a WebServer
- Don't use book.xml style sitemap solution
- Aggregate a page from xml files for each section, ie header, col1, col2, col3, and footer.
- Use a default xml file for each section

3 Solution

Layouting was taking from a description at [alistapart](#) ; it describes a column oriented layout by using css only.

The current implementation uses a flat URI. The base name of requested page is used inside the Cocoon sitemap for peeking xml documents from each of the section directories.

The response html page is aggregated by one xml-document from each of the section directories top-col-1, mid-col-1, mid-col-2, mid-col-3, mid-col-4, and bottom-col-1.

The directory layout:

Each section top-col-1, mid-col-1, mid-col-2, mid-col-3, mid-col-4, and bottom-col-1 holds documents of each section. A requested page is aggregated from the xml files having the same basename from each section directory. The sitemap resource `show-page` provides the aggregation.

If a section directory does not hold the requested xml document, the xml document index is used. The sitemap resource load-page provides this logical behaviour.

The directory `stylsheets` holds the xslt stylesheet.

The directory `resources` and its subdirectories `styles`, and `images` holds additional site resources.

Defaulting to default section document was implemented by an Action, and a Selector evaluating the Action result.

More specific the already existing `ResourceExistAction` is enhanced to a `ExtendedResourceExistsAction`. This action sets the sitemap parameter `resource-exists` to `true`, or `false`, depending if the requested source can be resolved, or not.

The act method of `ExtendedResourceExistAction`:

```
public Map act(Redirector redirector,
    SourceResolver resolver, Map objectModel,
    String source, Parameters parameters)
    throws Exception {
    String urlstring =
        parameters.getParameter("url", source);
    Source src = null;

    Map result = new HashMap();

    try {
        // try to resolve the source
        src = resolver.resolveURI(urlstring);
        src.getInputStream();

        // as no exception has been thrown assume that
        // the source exists, and is accessible
        result.put( RESOURCE_EXISTS_PARAM_NAME, "true" );

    } catch (Exception e) {

        // as an exception is thrown assume
        // that the source
        // can not be resolved, and does not exists
        getLogger().warn( "Resource " +
            String.valueOf(urlstring) +
            " does not exist, " +
            "set parameter " +
            RESOURCE_EXISTS_PARAM_NAME + " to false" );

        result.put( RESOURCE_EXISTS_PARAM_NAME, "false" );
    } finally {
```

CssColumnSite Application

```
        // do houskeeping release the resolved src
        resolver.release(src);
    }
    return result;
}
```

The selector `ParameterSelector` is used for evaluating the parameter `resource-exists`.

4 Sample Request Sequence

For example you request the page `csscolumnsite/faq.html`.

The sitemap matches the pattern `*.html`, invoking the resource `show-page`. The resource name `show-page` aggregates from its section parts the page. Each part strips the root element of the xml document file, and use the section name as its new part root element.

Finally the stylesheet `stylesheets/sit2html.xsl` replaces each part root name by the a `div id="section-name"` elements.

The resource `show-page` uses an internal pipeline for each section. Each section uses the resource `load-page` for loading the required xml document.

5 Configuration And More

Cocoon provides a `build/cocoon/webapp` directory. This site is put into the directory `csscolumnsite` under the `webapp` directory.

Thus you can access the index page of this site by `http://localhost:8888/cocoon/csscolumnsite/index.html` if you launched Cocoon's jetty instance.

Generating the static version of this site is done by using following ant task

```
<path id="webapp.classpath">
  <!-- add your cocoon-jars, classes etc. here
  -->
</path>
<target name="csscolumnsite"
  description="* build static csscolumnsite">

  <taskdef name="cocoon"
    classname="org.apache.cocoon.ant.CocoonTask"
    classpathref="webapp.classpath"/>

  <property name="build.dir"
    value="value=${cocoon.dir}/build"/>
```

```
<property name="contextDir"
  value="${build.dir}/cocoon/webapp"/>
<property name="workDir"
  value="${build.dir}/csscolumnsite-work"/>
<property name="destDir"
  value="${build.dir}/csscolumnsite"/>
<property name="configFile"
  value="${build.dir}/documentation/cocoon.xconf"/>

<cocoon
  contextDir="${contextDir}"
  configFile="${configFile}"
  workDir="${workDir}"
  destDir="${destDir}"
  targets="csscolumnsite/index.html"
  logLevel="WARN"
/>
</target>
```

The generated html documents are generated into the directory `build/csscolumnsite`. You can upload this directory, and its subdirectories to the destination webserver.

6 Adding A Document

You want to add an `mid-col-2` xml document, names `new-1.xml`.

Create a new xml file `new-1.xml` in the section directory `mid-col-2`. Use document element `<root>` as document root element. Use `xhtml` elements inside the root elements, save the xml document file.

More over you may want add a link to this new document by creating a link entry in the default `mid-col-1` xml document, ie `mid-col-1/index.xml`.

If you want to have a special `mid-col-1` document for your newly created document, create a new file `mid-col-1/new-1.xml` xml document, containing the content for the `mid-col-1` section.