ErrorHandling

Note: This document reflects error handling for Cocoon 2.0 series. Starting with 2.1, type="" attribute on <handle-errors/> element is deprecated and replaced with exception selector. Please refer to sample webapp – Vadim

Errors (e.g. Java Exception thrown during processing of a Pipeline) in Cocoon can be handled just like any other data: they can be fed into a pipeline for processing.

The Sitemap provides a means to describe how to handle processing errors at runtime, using the `map:handle-errors` element.

If a Sitemap doesn't include error handling instructions, then the exception will continue up the call stack and cause a ServletException that is reported by the servlet container.

Using an Error Handler

Each Pipeline can define its own processing for error handling as follows:

{{{<map:pipeline>

<!-- ... matchers ... -->

<map:handle-errors>
<!-- error handling instructions -->
</map:handle-errors>

</map:pipeline>
}}}

The error handling instructions can be any of the usual Sitemap ErrorGenerator] *except* a Generator. This is because Cocoon has a special [http://xml.apache.org/cocoon/userdocs/generators/error-generator.html which is used to feed details of an exception, as SAX events, into the error handling pipeline. The ErrorGenerator documentation includes a description of the XML that is created by the ErrorGenerator.

This means that you can create a user-friendly error page by adding a Transformer and Serializer to the `<map:handle-errors/>` element. The default Cocoon Sitemap does this as follows:

{{{<map:handle-errors>
<map:transform src="stylesheets/system/error2html.xsl"/>
<map:serialize status-code="500"/>
</map:handle-errors>
}}}

Note that you can look at the `error2html.xsl` stylesheet as an example of how to process error documents using XSLT, and that the (default) HTML serializer is being instructed to return an HTTP status code of 500 (Server Error) to the users browser.

Per Pipe Handlers

Looking through the default Cocoon Sitemap you can see that it's mostly a single large Pipeline with a single error handler. So it seems to make sense to follow this pattern if you have a generic way of handling errors.

However if you want to have specific error reports from particular kinds of processing, you should create separate Pipelines which can have their own error handling instructions. E.g:

{{{<map:pipeline>
<!-- XSLT transform pipe -->
<map:handle-errors>
<!-- handle errors for this pipeline
</map:handle-errors>
</map:pipeline>

<map:pipeline>
<!-- Reading transform pipe -->
<map:handle-errors>
<!-- custom handler for second pipeline
</map:handle-errors>
</map:pipeline>
}}}

Reader comments

Per Error Per Pipe Handlers

I'd been trying to figure error handling out for some time now and have discovered another method for declaring custom errors based on type. I tried the example above, and Cocoon responded with a "Cocoon Confusion" error. After some digging through the sample files, I discovered the method below:

The following code goes after the <map:match />; section(s) in your pipeline(s). (I found the example in the sitemap in the "sub" directory under the default Cocoon installation.)

```
<map:handle-errors type="500">
     <map:transform src="stylesheets/error2html.xsl"/>
     <map:serialize/>
</map:handle-errors>
```

This method allows you to specify multiple custom error pages for multiple pipelines based on the error type.

– BryanRobison

Error type and ResourceNotFound exception

There seems to be only two types of error notification (as explained in the thread Beautify 404) :

- Type **500**: every exceptions except `ResourceNotFound` and `ConnectionReset` (special status)
- Type **404**: `ResourceNotFound` (eg. permission denied, file not found, etc.)

You can have up to two <map:handle-errors> elements - one of every type. By default (cocoon-2.0.4), no handler is defined for `ResourceNotFound` (ie. `type="404"`), hence the different error page which is produced by the Servlet Container.

Typical problems may look like this: no matter how hard you customize your error2html stylesheet, you still get the same plain error page produced by cocoon servlet (ex: when missing a file needed by a <map:generate>)

The sitemap fragment below shows the use of the two different types in one pipeline

```
  <map:pipeline>
    <map:match pattern="test/**">
      <map:mount check-reload="yes" src="oreilly/sitemap.xmap" uri-prefix="test"/>
    </map:match>
    <map:handle-errors type="500">
     <map:transform src="stylesheets/system/error-2-html.xsl"/>
     <map:serialize/>
    </map:handle-errors>
    <map:handle-errors type="404">
     <map:transform src="stylesheets/system/error404-2-html.xsl"/>
     <map:serialize/>
    </map:handle-errors>
  </map:pipeline>
```

Remember that when a <map:handle-errors> is not found for the type of exception you want to process in the current pipeline, cocoon will try to find one in the parent sitemap (if using sub-sitemap). If no such handler is found, the exception will be handeld by the Cocoon Servlet !

– SergeColin

Error Handling in Cocoon 2.1

Changes made by Sylvain Wallez are outlined and debated in this thread. The first message includes details about how backwards compatibility with Cocoon 2.0.x is preserved.

– MarkLeicester

How to get Exception Object

if you want to display the original error message in your own xsp ...

```
<xsp:logic>
// dump error to stderr so we can fix it. /te

System.err.println("---UNCAUGHT EXCEPTION IN SITEMAP");

org.apache.cocoon.components.notification.Notifying  notification  =
(org.apache.cocoon.components.notification.Notifying)objectModel.get(org.apache.cocoon.Constants.
NOTIFYING_OBJECT);

System.err.println(notification.getDescription());
java.util.Map m = notification.getExtraDescriptions();
if (m!=null)
{
        Object t = m.get("stacktrace");
        if (t!=null)
          System.err.println(t);
}
System.err.println("---");
</xsp:logic>
```

(works for cocoon 2.1rc1)

– TomEicher 25.08.03

How to use <xsl:message>?

The notifying generator and the XSP above don't show the original error message if it comes from a `<xsl:message>` in a XSL stylesheet. This always gives the message

```
org.apache.cocoon.ProcessingException: Failed to execute pipeline.: java.lang.RuntimeException: Stylesheet
directed termination
```

Does anyone know a way to get the original message?

– NicoVerwer 02.03.04