CInclude Transformer (2.1 legacy document)

Table of contents

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.

Table of contents

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.

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. CInclude Transformer

This transformer includes XML in the current stream and acts therefore as a kind of (dynamic) content aggregation. Two forms are supported by the transformer: one verbose and flexible approach, and a simple approach. We will first discuss the simple approach and the more flexible is mentioned in the next chapter. In addition the cinclude transformer provides a caching mechanism (for the simple include form).

This transformer triggers for the element include in the namespace "http://apache.org/cocoon/include/1.0". The src attribute contains the url which points to an xml resource which is included instead of the element. With the attributes element, ns and prefix it is possible to specify an element which surrounds the included content.

- Name : cinclude
- Class: org.apache.cocoon.transformation.CIncludeTransformer
- Cacheable: no.

A simple example might help to use the CIncludeTransfomer effectively:

Add the CIncludeTransformer to the components in your sitemap.xmap

... <map:components> ... <map:transformers default="xslt"> ... <map:transformer name="cinclude" src="org.apache.cocoon.transformation.CIncludeTransformer"/> ...

Next define in your pipeline to use the CIncludeTransformer

<map:match pattern="cinc/simple-cinc"> <map:generate src="cinc/simple-cinc.xml"/> <map:transform type="cinclude"/> <map:transform src="stylesheets/page/simple-page2html.xsl"/> <map:serialize/> </map:match>

In this example pipeline it assumed that simple-cinc.xml contains the include element. Beside defining the include element it defines the namespace URI "http://apache.org/cocoon/include/1.0". This helps the CIncludeTransformer to find the tag to get replaced by the xml content referenced via the src attribute. The simple-cinc.xml may look like this:

<?xml version="1.0" encoding="UTF-8"?> <page xmlns:cinclude="http://apache.org/cocoon/include/1.0"> <title>Hello</title> <content> <para>This is my first Cocoon page!</para> <cinclude:include src="include.xml" element="included"/> </content> </page>

Next you should define the include.xml file which is included. A simple include.xml might look like this:

<?xml version="1.0"?> I am included by CIncludeTransformer. I come from "include.xml".

Now finally we have everything put together the xml content after the CIncludeTransformer processing will look like this:

<?xml version="1.0" encoding="UTF-8"?> <page xmlns:cinclude="http://apache.org/cocoon/include/1.0"> <title>Hello</title> <content> <para>This is my first Cocoon page!</para> <included> I am included by CIncludeTransformer. I come from "include.xml". </included> </content> </page>

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.

2. Including External XML (simple)

One feature of the cinclude transformer (this is currently not supported by the caching cinclude transformer) is including XML from external sources, e.g. files or from an HTTP server. The cinclude:includexml tag starts including of XML:

```
<cinclude:includexml> <!-- Include XML from HTTP server -->
<cinclude:src>http://external.news.com/flashnews.xml</cinclude:src> </cinclude:includexml>
```

This would be a simple way of "get"ting XML data from an external site. Using this method it is also possible to pass parameters in the url - just as you would in a "get" sent from a browser.

```
<cinclude:includexml> <!-- Include XML from HTTP server -->
<cinclude:src>http://external.news.com/flashnews.xml?id=1234&myname=matthew</cinclude:src>
</cinclude:includexml>
```

If the external XML is not valid or not available, the transformer signals an error to the pipeline and the document containing the include command is not available.

For this purpose the ignoreErrors attribute can be used:

```
<cinclude:includexml ignoreErrors="true"> ... </cinclude:includexml>
```

3. Including External XML (advanced)

The above section shows you how to include XML data from an external source such as an HTTP server using the simple "get" method supplied in the HTTP protocol. For more advanced uses you will wish to be able to send "Post" or other HTTP methods to the server. In addition you may want to actually send XML data to the server - just as you would using an HTML form. The format of this resource is slightly more complicated:

```
<?xml version="1.0"?> <data xmlns:cinclude="http://apache.org/cocoon/include/1.0">
<cinclude:includexml> <cinclude:src>http://itsunshine/tamino/blah</cinclude:src>
<cinclude:configuration> <cinclude:parameter> <cinclude:name>method</cinclude:name>
<cinclude:value>POST</cinclude:value> </cinclude:parameter> </cinclude:configuration>
<cinclude:parameters> <cinclude:parameter> <cinclude:name>message</cinclude:name>
<cinclude:value>Hi there</cinclude:value> </cinclude:parameter> <cinclude:parameter>
<cinclude:name>_Process</cinclude:name>
<cinclude:value><name>matti</name><age>36</age></cinclude:value>
</cinclude:parameter> </cinclude:parameters> </cinclude:includexml> </data>
```

Lets look at the tags. The tag cinclude:src defines the address of the resource we want to access and then comes a list of (optional) connection-specific parameters (enclosed in the cinclude:configuration tag). In this example the HTTP-method ("POST") is passed into the connection. The format of these parameters is discussed next.

Then comes the list of parameters we wish to pass into the function. Each parameter defined has a name and a value. The value can either be text or XML.

The format of the parameters is the same as for the connection configuration.

4. Caching

This transformer includes XML in the current stream and acts therefore as a kind of (dynamic) content aggregation. However, the included content might be very big or either it might take a lot of time to fetch the content. If, in those cases, your content does not change too frequently, you can turn on

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.

caching for these contents.

To turn on caching, this transformer triggers for the element cached-include in the namespace "http://apache.org/cocoon/include/1.0/caching". The src attribute contains the url which points to an xml resource that is included instead of the element. It is possible to mix the cached-include and the include element, so only parts are cached and others are not.

A simple example might help to use the caching effectively:

First define your pipeline to use the CIncludeTransformer with caching turned on; you turn on caching by setting the expires parameter to a value greater than 0. The exact meaning of this parameter is explained below.

```
<map:match pattern="cinc/simple-cinc"> <map:generate src="cinc/simple-cinc.xml"/>
<map:transform type="cinclude"> <map:parameter name="expires" value="600"/>
</map:transform> <map:transform src="stylesheets/page/simple-page2html.xsl"/>
<map:serialize/> </map:match>
```

In this example-pipeline it is assumed that simple-cinc.xml contains the cached-include element. Beside defining the element it uses the namespace URI "http://apache.org/cocoon/include/1.0". This helps the transformer to find the tag to get replaced by the xml content referenced via the src attribute. The simple-cinc.xml may look like this:

```
<?xml version="1.0" encoding="UTF-8"?> <page
xmlns:cinclude="http://apache.org/cocoon/include/1.0"> <title>Hello</title> <content>
<para>This is my first Cocoon page!</para> <cinclude:cached-include
src="http://server/document1.xml"/> <cinclude:cached-include
src="http://server/document2.xml"/> </content> </page>
```

Now finally we have everything put together the xml content after the CIncludeTransformer processing will look like this:

```
<?xml version="1.0" encoding="UTF-8"?> <page
xmlns:cinclude="http://apache.org/cocoon/include/1.0"> <title>Hello</title> <content>
<para>This is my first Cocoon page!</para> <document1> CONTENT OF document 1
</document1> <document2> CONTENT OF document 2 </document2> </content> </page>
```

So, of course even with caching turned on, this transformer acts like the usual cinclude transformer. But as you can see from the example above, you can define an expires value. The fetched content is cached for the duration of this value; in the example above the content is cached for 10 minutes. So, if during the next 10 minutes after the first time this pipeline was processed, someone else requests this pipeline, the content is not fetched again from a distant server (or whereever the content is stored). It is directly delivered from the cache. When the 10 minutes have expired, the next time the pipeline is requested, the content is fetched again and stored in the cache for the next 10 minutes.

You can fine tune the behaviour of the transformer with several parameters.

The expires parameter defines the expiration date of the content in seconds from the time the pipeline is requested.

Usually the content is cached in the common store, but you can also define a writeable/modifiable source with the "source" parameter, e.g. "file:/c:/temp". Then the cached content is written into this directory.

With the optional purge set to true the cache is purged which means the cached content is regarded as invalid nevertheless if it has expired or not.

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.

With the optional parameter parallel the various included contents are processed (included) in parallel rather than in a series.

With the optional parameter preemptive set to true a pre-emptive caching is activated. When a resource is requested with pre-emptive caching, this transformer always attempts to get the content from the cache. If the content is not in the cache, it is of course retrieved from the original source and cached. If the cached resource has expired, it is still provided. The cache is updated by a background task. This task has to be started beforehand.

Complete Example:

```
<map:match pattern="cinc/simple-cinc"> <map:generate src="cinc/simple-cinc.xml"/>
<map:transform type="cinclude"> <map:parameter name="expires" value="600"/>
<map:parameter name="purge" value="false"/> <map:parameter name="parallel"
value="true"/> <map:parameter name="preemptive" value="false"/> <map:parameter
name="source" value="file:/c:/temp"/> </map:transform> <map:transform
src="stylesheets/page/simple-page2html.xsl"/> <map:serialize/> </map:match>
```

5. Configuration

Besides the usual transformer configuration, this transformer requires some components. You have to add the following lines to the cocoon.xconf:

```
<component
class="org.apache.cocoon.transformation.helpers.DefaultIncludeCacheManager"
role="org.apache.cocoon.transformation.helpers.IncludeCacheManager" logger="test"> <!--
Specify this only if you use preemptive-caching --> <parameter name="preemptive-loader-url"
value="http://localhost:8080/cocoon/loader"/> </component>
```

If you want to use preemptive caching, you have to specify a URI inside Cocoon that contains the preemptive-loader action. This pipeline is automatically called, when preemptive loading is actived and requried. It loads the content in the background.

First you have to define the action:

```
... <map:components> ... <map:actions> ... <map:action name="preemptive"
src="org.apache.cocoon.transformation.helpers.PreemptiveLoaderAction"/> ...
```

Then you must define a pipeline containing the action. This is the pipeline that has to be configured in the cocoon.xconf:

```
<map:match pattern="loader"> <map:act type="preemptive"></map:act> </map:match>
```

1. Comments

add your comments

Copyright © 1999-2005 The Apache Software Foundation. All rights reserved.