Modules (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. Introduction

Many sitemap components serve a purpose regardless how the input is obtained. Still, to provide a wide range of components to quickly get you up to speed, variants for different inputs exist. Modules allow to create generic components and plug-in input or output later.

This document will explain how modules work and how to make use of them. If you plan on writing your own modules, it is highly recommended to read Developing With Apache Avalon. It is a very good description of the underlying rationale and principles.

2. Types of Modules

Currently, three different types of modules exist: Input modules provide means to enumerate attributes and to retrieve them, output modules allow storing of data and exhibit transaction like semantics, database modules encapsulate different mechanisms for auto increment columns of various database management systems. Please refer to the javadoc documentation of these interfaces.

Input modules are modelled after request attributes. The main difference is, that every method takes two additional arguments, the request object and a configuration object. The configuration object is used to allow arbitrarily complex instructions for the input module. Apart from that, input modules are more or less a drop-in replacement.

Output modules are again very similar to using request attributes. Basically, they provide a method to set an attribute to a value. Again, a request and a configuration object is the only change to request attributes. A fundamental difference is, however, that output modules should exhibit transactional behaviour. Thus setting an attributes implicitly starts a transaction that must be ended by calling rollback or commit. Only if the transaction is completed by calling commit, the values set should be visible. This is needed e.g. by the database actions.

Database modules, actually named AutoIncrementModule, contains configuration information how to retrieve a value for an auto increment column. It is possible to obtain the value before inserting a row, while inserting as part of the SQL or after successful insert. If the value is obtained before inserting, it can be generated externally. Currently, supported database management systems include HSQL, Informix, MySQL, and querying the database for the current max value.

3. Using modules

Using any of these modules requires a two step setup process. Step one has already been done for your for all modules that come with Apache Cocoon. Exception to this rule are the auto increment modules: only the HSQL module is already setup.

3.1. Step 1: Making a new module known to Apache Cocoon

Like other core components of Apache Cocoon, modules are declared in cocoon.xconf. There are already too many to list here.

```
<input-modules> <component-instance name="request"
class="org.apache.cocoon.components.modules.input.RequestParameterModule"/>
<component-instance name="attribute"
```

```
class="org.apache.cocoon.components.modules.input.RequestAttributeModule"/>
<component-instance name="URI"
class="org.apache.cocoon.components.modules.input.RequestURIModule"/>
<component-instance name="context"
class="org.apache.cocoon.components.modules.input.RequestContextPathModule"/>
<component-instance name="header"
class="org.apache.cocoon.components.modules.input.HeaderAttributeModule"/>
<component-instance name="session"
class="org.apache.cocoon.components.modules.input.SessionAttributeModule"/>
<component-instance name="date"
class="org.apache.cocoon.components.modules.input.DateInputModule"/>
<component-instance name="defaults"
class="org.apache.cocoon.components.modules.input.DefaultsModule"> <input-module
name="request"/> <values> <skin>defaultSkin</skin>
<base-url>http://localhost:8080/cocoon</base-url> </values> </component-instance>
</input-modules> <output-modules> <component-instance name="attribute"
class="org.apache.cocoon.components.modules.output.RequestAttributeOutputModule"/>
<component-instance name="session"
class="org.apache.cocoon.components.modules.output.SessionAttributeOutputModule"/>
</output-modules> <autoincrement-modules> <component-instance name="auto"
class="org.apache.cocoon.components.modules.database.HsqlIdentityAutoIncrementModule"/>
<!-- <component-instance name="auto"
class="org.apache.cocoon.components.modules.database.ManualAutoIncrementModule"/>
<component-instance name="auto"
class="org.apache.cocoon.components.modules.database.IfxSerialAutoIncrementModule"/>
<component-instance name="auto"
class="org.apache.cocoon.components.modules.database.MysqlAutoIncrementModule"/> -->
</autoincrement-modules>
```

The above snippet declares a number of modules. After this, the modules are accessible through the given name. Thus, when an input-module is expected, it is sufficient to give the name of a module, like header.

For the auto increment modules only one is declared as the name "auto" has special meaning to the modular database actions. If more than one is needed at the same time, the configuration of the database actions needs to explicitly specify which one to use.

3.2. Step 2: Use it

The following alternatives for using modules exist:

3.2.1. Step 2a: Use it as sitemap variable

Input modules can be used in a sitemap almost like a sitemap variable. If the variable name contains a colon (":"), the preceeding string is used as the name of the module to use and the trailing string is passed to the module. The expression is replaced with the string value returned from the module.

```
<map:transform src="resources/stylesheets/{../skin}.xsl"/>
```

The above example uses the variable skin declared e.g. by an action for the stylesheet to apply to the page. The example below uses an input module instead. The way this module was declared above allows to override the skin with a request parameter named "skin".

```
<map:transform src="resources/stylesheets/{default:skin}.xsl"/>
```

Some of the input modules are JXPath-enabled, so you can use XPath expressions to access values (see Input Modules sample for details). The following example demonstrates the use of XPath function with system-property module.

<map:parameter name="users-home-base"
value="{system-property:substring-before(user.home, user.name)}"/>

3.2.2. Step 2b: Use it on an XSP

The input logicsheet allows easy use of InputModules from an XSP. Currently, it provides tags for getting one value, an array of values, and an Iterator for a Collection of parameter names.

<?xml version="1.0" encoding="ISO-8859-1"?> <xsp:page language="java" xmlns:xsp="http://apache.org/xsp" xmlns:input="http://apache.org/cocoon/xsp/input/1.0"> <page> <title>Testing InputModules</title> Parameter name=<input:get-attribute module="request-param" as="string" name="module" default="John Doe"/>;   Parameter cars=<input:get-attribute-values module="request-param" as="xml" name="car"/>; </page> </xsp:page>

3.2.3. Step 2c: Have sitemap components use a module

This depends on the component that is to be used. As an example the CachingWildcardMatcher requires to set the input-module on declaration.

<map:matchers default="wildcard"> <map:matcher name="cached-uri" src="org.apache.cocoon.matching.modular.CachingWildcardMatcher"> <input-module name="URI"/> </map:matcher> </map:matchers>

By replacing the input module name with any of the other declared input modules, this matcher can be used to match e.g. on session attributes, request headers or even dates!

1. Comments

add your comments