IntegrateAServlet

This How To describes the following:

1. invoking a servlet that runs outside Cocoon (e.g. in a separate webapp) and process the results in a Cocoon pipeline.
2. add an existing custom servlet to Cocoon, and similarly process its output

This is a common requirement if you have an existing servlet that generates XML data using custom logic and you'd rather not have to turn it into a Generator or an XSP page.

Invoking A Servlet

Invoking a servlet, or indeed any other kind of web accessible resource (e.g. a CGI app) from a Cocoon Pipeline is simple: you just provide its URL to a Generator.

If you servlet (or CGI) produces XML then just use the default XML generator:

```
<map:pipeline match="...">
  <map:generate src="http://my.server.com/path/to/my/servlet"/>
  <!-- rest of pipeline to process results -->
</map:pipeline>
```

If your servlet produces HTML then you may want to use the HTML Generator instead, as this uses JTidy to produce well-formed output. Simply add a `type="html"` attribute to the above example.

If you need to pass parameters through to the servlet, then you can use the Request Parameter Action as follows:

```
<map:pipeline match="...">
  <map:act type="request">
    <map:parameter name="parameters" value="true"/>
    <map:generate src="http://my.server.com/path/to/my/servlet{requestQuery}"/>
  </map:act>
  <!-- rest of pipeline to process results -->
</map:pipeline>
```

Notice that the query string is passed through to the servlet by adding the `requestQuery` sitemap parameter, created by the Action, to the end of the URL.

Adding a Servlet to the Cocoon Webapp

Add the Classes

Firstly you need to ensure the classes for your application are available from the `CLASSPATH`. You can either put the classes in `$COCOON_HOME/WEB-INF/classes` or package it up as a jar file and place it in `$COCOON_HOME/WEB-INF/lib`

Declare the Servlet

You then need to declare that the servlet should be loaded into the Cocoon webapp by editing `$COCOON_HOME/WEB-INF/web.xml`. You'll need to add XML similar to this:

```
<servlet>
 <!-- name used to refer to servlet -->
 <servlet-name>servletName</servlet-name>
 <!-- fully-specified class name of servlet -->
 <servlet-class>com.server.my.ServletClass</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>servletName</servlet-name>
  <!-- the URL path at which the servlet is mounted -->
  <url-pattern>/internal/myServlet</url-pattern>
</servlet-mapping>
```

This is all standard Java web application config, so consult a reference if you have problems.

When you reload the Cocoon webapp (e.g. restart Tomcat) your new servlet will be accessible from `/internal/myServlet`.

Integrating the Servlet

Integrating the servlet into your pipeline then follows the procedure outlined at the start of the document. For example you might reference it as:

```
<map:pipeline match="some/url/path/foo.xml">
  <map:generate src="/internal/myServlet"/>
  <!-- rest of pipeline to process results -->
</map:pipeline>
```

The key thing to understand is that your servlet has to be mounted at a URL (configured by `web.xml`) before it can be called from a pipeline.

This How To is basically an edited form of an exchange that took place between Kavitha Ramesh and Everett (Skip) Carter on the `cocoon-users` mailing list. kavitha confirmed that the advice was correct.

The only changes have been to present the information in a more readable form, and some tweaking of the examples.