LepidoSitemapDebugger

Lepido Sitemap Debugger Contribution

Introduction

The Sitemap Debugger offers an Eclipse based GUI to trace the processing of the Cocoon sitemap. It consists of two main parts: An Eclipse plugin and a cocoon extension. For Cocoon 2.1.8dev there is a patched version of the treeprocessor which adds debugging support. Cocoon 2.2 provides a debugging interface which makes it easy to add a the debugger. A default implementation, which works with the provided debugger, is included in the profiler-block. The debugging interfaces should be backported to Cocoon 2.1.8 if they are finished to ease the development.

Current Functions

- request based tracing of the sitemap
- display sitemap variables
- supported sitemap elements:
  - matcher
  - generator
  - transformer
  - serializer
  - ...
- supported Cocoon Versions: 2.1.8dev and 2.2dev
- supported Eclipse Verison: 3.1

ToDo-List

- add support for the missing sitemap elements
- add support for chanding sitemap variables
- discuss if it usefull to port it to the Eclipse debugging infrastructure
- dicsuss new features
- backport the sever-side debugging-interaces to Cocoon 2.1.8dev if they are finished
- extend the documentation

Dev Info

Eclipse Client

The basic function of the Eclipse debugging client is rather easy: read a message from the server, parse the message and update the views, send the next command to the server. But the code in fact is in some cases not so easy to understand 🙂

Here is a short overview what the classes do:

package org.eclipse.lepido.debugger:

DebuggerPlugin: Main plugin class, provides some functionality for logging. BrowserControl: Helper to launch an external browser (should be replaced with the SWT browser stuff).

package org.eclipse.lepido.debugger.sitemap

DebugView: Main view of the Plugin, displays the sitemap and provides the controls to start, stop and configure the debugger

MessageBroker: Provides functions to process the incoming messages

Perspective: The Eclipse perspective

RequestThread: Reads a stream from an URL, used if you don't want to view the result in the browser

Sitemap: Holds the sitemap as a dom4j-object

SitemapContentProvider: provider for the treeview to display the sitemap

SitemapLabelProvider: provider for the treeview to display the labels

SocketControlerThread: Thread to start and stop the server, to read and send messages

StartDialog: Dialog that pops up if you want to start the debugger

StatementMessage: Represents a <sitemap-element> message

StatementPropertySheetEntry: PropertySheetEntry for the information values of a statement-message

StreamView: This view displays the processd streams

ViewRunnable: Forwards method calls from the SocketControlerThread to the DebugView

XMLMessage: Represents a XML-message from cocoon

package org.eclipse.lepido.debugger.xmlsourceviewer:

This package contains a simple source viewer to display nice colored xml in the StreamView

Cocoon Extension

2.1.8dev

The Cocoon extension for 2.1.8 is a patched TreeProcessor with some modified sitemap classes. Have a look at the treeprocessor package and search for references of the Debugger class.

2.2dev=

Cocoon 2.2 provides a debugger interface for the TreeProcessor. A default implementation is provided in the profiler block. The source can be found in the org.apache.cocoon.profiler.debugging package (Debuger.java and RemoteDebuggingSitemapExecutor.java)

Protocol

The debugmode is invoked via a special request parametert: http://server/cocoon/test?remote-debug=host:port

If the connection is established Cocoon sends the sitemap with the first message:

```
<message>
   <sitemap src="..">
           <!-- Here is the sitemap -->
   </sitemap>
</message>
```

The following message is send for every processd sitemap element:

```
<message>
        <sitemap-element>
           <information>
           <!-- Result from the previous element -->
                   <values>
                           <value name="...">___</value>
                  </values>
           </information>
            <statement type="..."> <!-- type is match, generate... -->
                   <!-- type specific information, one element for each attribute: -->
                   <src>___</src>
                   ...
                   <parameters>
                           <parameter name="..">...</parameter>
                  </parameters>
      </statement>
        </sitemap-element>
</message>
```

If a stream is processd (generator or transformer) the result will be send to the client:

```
<message>
   <stream>
   ...
   </stream>
</message>
```

Finish messagen::

```
<message>
   <finished/>
</message>
```

The client will only send short status messages:

```
<message>
   <status>0</status>
</message>
```

0 = next message
-1 = cancel current operation