GT2005HackatonDay1Notes

Cocoon 2.2 Internals

Cocoon internals

Carsten Ziegeler and Reinhart Potz are walking through the new Cocoon 2.2 code.

See the SVG picture in cocoon_startup.svg.

First is the Environment/Servlet is setup.

It creates the BootstrapEnvironment. After that the CoreUtil is started and it sets up the cocoon core. It instantiates the avalon context and loads all settings. The core also instanties the avalon component system. The serviceManager is created. It loads the cocoon.xconf.

The serviceManager is part of the avalon components. It uses the cocoon object. The cocoon object handles the request processing.

They start talking about the ServletBootstrap environment. It basically creats a wrapper around the servlet context.

The next thing is creating the coreUtil. It's the core class for setting up cocoon.

After that cocoon is created with the createCocoon function. It sets up the logging system.

Cocoon is created with the new Cocoon object. Then the core is created.

Settings object

The settings object is new in Cocoon 2.2 It stores all directories etc. Most of them come from web.xml in Cocoon 2.1

Starting with 2.2 it's possible to use propertyfiles The main one is WEB-INF/cocoon-settings.properties

It scans WEB-INF for all properties files

There is also a properties directory from where you can get additional propertie files.

The idea of the settings object is to move away from the avalon context.

The other nice thing of the settings is that it can store every property.

Currently each block can have it's own property file.

For each block there's a properties file

You can set cocoon to use a custom property file as a system property.

CoreUtil object

Daniel suggests to use servlet interfaces instead of our own interfaces

RequestFactory object

The request factory creates a wrapper around each request.

Is a wrapper around each request to handle fileuploading

BootstrapEnvironment CoreUtil
ServiceManager

Important for cocoon is:

- the core.

Into debugging mode

Coreutil gets a new Cocoon instance

For each request, the following objects are created:

Environment
--> Request object
--> Response object

Cocoon.java

process method is the main method

It starts by calling the environment.startingProcessing

EnvironmentHelper.java is an environment helper class

Then it fires enterProcessor which pushes the environmentinfo on the stack

You can hook up Request listeners

Treeprocessor

In early 2.0 versions, the sitemap used to be compiled to Java using the same engine as XSP But reloading large sitemaps was very slow and Sylvain introduced the treeprocessor

Sylvain will explain something about the tree processor. Rewrite the sitemap engine based on a navagation tree. Tree of object.

- First step is building the tree
- Second step is execute it.

- Tree processing loadbuilding
- Tree processing node

Build tree of object and goes down the DOM.

Stays alive for the current request. Rebuild after finished all current requests

For each statement in the sitemap there's a class.

As an example we dive into the GenerateNode.java

changeContext is important. Each time we enter the sitemap we change the context.

When you mount a sitemap, we change the context of the sourceresolver.

ConcreteTreeProcessor has a counter of current request. If 0 then dispose the old sitemap object

each processing node has an invoke method. has the sitemap parameters and the pipeline object. then execute the executor that does the work

new feature pipeline hints

Generate is the first one that sets up the pipeline. Get this object van the pipeline selector depending on the type of the attribute.

and then set on the pipeline the generator.

Then get current generator from the service manager

In your component you can get the location of the pipeline

SerializeNode.java

Check for either a generator or a reader

The pipeline is requested from the context. The source, parameters and mimetype is requested. And finally the serializer is set.

Each sitemap and subsitemap has an owner: the ServiceManager

Drawing will be added later. They are created by [JR]

The sitemap asks the ServiceManager for

a FileGenerator an XSLT Transformer an HTML Serializer etc

A sub-sitemap will first look up a component using its own ServiceManager. If it is not found, the lookup is deferred to its parent sitemap's ServiceManager, etcetera... See the SVG picture in subsite_component_lookup.svg.

Building wil start with creating the serviceManager with the map components/sitemap components.

In 2.2 we have a classpath for each sitemap.

ServiceManager gets a classloader from...

When all is setup, it starts the process method of the pipeline

First setup of the generator is called with the parameters specified in the sitemap.

Finally a serializer is setup as well.

The pipeline is setup and each components has all information it needs

Regardless of any Selectors, Matchers etc, the pipeline merely consists of generator, transformer, transformer, transformer, serializer.

Selectors

What actually happens with a selector? And actions?

Processed from top to bottom.

Sitemap is build once. Actions and selectors are handled directly.

Why is it not done in SAX?

Pier said:....

Nico Verwer: Need a picture of the three stages. This has been made by Helma and Nico, here are the SVGs.

request_processing.svg

request_processing_sub_sitemap.svg

request_processing_cocoon_protocol.svg

Actions are started and finished before processing the content?

If you have a pipeline using an internal cocoon source two pipelines are build. This happens when the first generator uses the source resolver to fetch the content. So they are not created one after another.

Three points:

Read the sitemap and build the graph Walk through the sitemap and assemble components for that specific request.

Then generator.generate();

CachingPipeline

Components are connected by prev and next, consumer and producer

XMLTeePipe gets called in the middle to handle caching

If request is found in the cache it sends 304 (not modified) in the HTTP response.

The serializer catches all the SAX events generated by the generator.

Each time the cocoon:// protocol is used it will invoke the buildPipeline function.

InputModules are called when the component is added to the pipeline. The variableResolver will handle the input modules. So the cocoon pipeline called by an inputModule will be handled before the actual generate is called.