Parent Service Manager (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. Parent Service Manager

When using Apache Cocoon it is sometimes neccessary to obtain components from other sources than the user.roles file, or preferable to have a common component manager for several web applications.

The pattern chosen for Cocoon is the dynamic loading of a service manager class. The initialization parameter parent-service-manager in web.xml specifies a class that will be loaded, instantiated and used as a parent service manager for Cocoon's serivce manager.

The recommended procedure is for the class, when it is initialized, to create a delegate in the form of an CocoonServiceManager, configure it by looking up a Configuration object via JNDI, and delegate any requests to it.

In order to provide a way to pass parameters to the parent service manager class (the class specified in parent-service-manager), Cocoon will instantiate the class via the constructor that takes a single String argument, passing anything to the right of the first '/' in the parameter value to the constructor. Subsequently Cocoon examines whether the class implements org.apache.avalon.framework.logger.LogEnabled and/or org.apache.avalon.framework.activity.Initializable and calls setLogger and/or initialize, as appropriate. The instance is then used as a parent service manager.

Since that didn't make much sense in itself, let's look at the sample.

The goal is to define a component that can give us the time of day and let it be managed by a parent service manager.

So, first we need to put a Configuration object into JNDI, and then grab that object, use it to configure an CocoonServiceManager, and pass on any requests to that manager.

1.1. Step 1: Creating a configuration object

We'll do this the quick and dirty way. The static initializer of a class will create a Configuration instance with a single role and bind it to org/apache/cocoon/samples/parentcm/ParentCMConfigration.

The following code was taken from org/apache/cocoon/samples/parentcm/Configurator.java

```
public class Configurator { static { try { // // Create a new role. // DefaultConfiguration config = new DefaultConfiguration("roles", ""); DefaultConfiguration timeComponent = new DefaultConfiguration("role", "roles"); timeComponent.addAttribute("name", Time.ROLE); timeComponent.addAttribute("default-class", TimeComponent.class.getName()); timeComponent.addAttribute("shorthand", "samples-parentcm-time"); config.addChild(timeComponent); // // Bind it - get an initial context. // Hashtable environment = new Hashtable(); environment.put(Context.INITIAL_CONTEXT_FACTORY, MemoryInitialContextFactory.class.getName()); initialContext = new InitialContext(environment); // // Create subcontexts and bind the configuration. // Context ctx = initialContext.createSubcontext("org"); ctx = ctx.createSubcontext("apache"); ctx = ctx.createSubcontext("cocoon"); ctx = ctx.createSubcontext("samples"); ctx = ctx.createSubcontext("parentcm"); ctx.rebind("ParentCMConfiguration", config); } catch (Exception e) { e.printStackTrace(System.err); } } }
```

To make sure the static initializer runs we make Cocoon force-load the class by making a change to the web.xml file:

<init-param> <param-name>load-class</param-name> <param-value> <!-- For IBM WebSphere: com.ibm.servlet.classloader.Handler --> <!-- For Database Driver: --> @database-driver@ <!-- For parent ServiceManager sample: This will cause the static initializer to run, and thus the Configuration object to be created and bound. --> org.apache.cocoon.samples.parentcm.Configurator </param-value> </init-param>

1.2. Step 2: Write the service manager

Now that the configuration object is sitting there waiting for us, let's craft the component manager. Please see the file org/apache/cocoon/samples/parentcm/ParentServiceManager.java for an example. It is too much to paste in here.

1.3. Step 3: Tell Cocoon to use the service manager

Change the web.xml file to:

<init-param> <param-name>parent-service-manager</param-name> <param-value>org.apache.cocoon.samples.parentcm.ParentServiceManager/(remove this line break) org/apache/cocoon/samples/parentcm/ParentCMConfiguration</param-value> </init-param>

Cocoon will now do the following: First, it will split the parameter value at the first slash, in this case ending up with the strings "org.apache.cocoon.samples.parentcm.ParentServiceManager" and "org/apache/cocoon/samples/parentcm/ParentCMConfiguration". The first string is the class to instantiate. The second is the parameter that will be passed to the constructor.

Next, Cocoon loads the component manager class and uses reflection to find a constructor that will accept a single String argument. Upon finding one, it instantiates the class in a manner similar to:

ServiceManager cm = new org.apache.cocoon.samples.parentcm.ParentServiceManager("org/apache/cocoon/samples/parentcm/ParentCMConfiguration");

After this Cocoon checks whether the parent service manager class implements Initializable and/or LogEnabled. Since the ParentServiceManager class implements both, Cocoon does the following (with simplification):

((LogEnabled) cm).enableLogging(logger); ((Initializable) cm).initialize();

Finally, the instance is used as parent service manager of Cocoon's own service manager.

1.4. Step 4: Use the component

Cocoon components can now use the ServiceManager given to them by Cocoon to look up the component managed by the parent service manager:

The following code was taken from org/apache/cocoon/samples/parentcm/Generator.java

public void setup(SourceResolver resolver, Map objectModel, String src, Parameters par) throws ProcessingException, SAXException, IOException { Time timeGiver = null; try { timeGiver = (Time) manager.lookup(Time.ROLE); this.time = timeGiver.getTime (); } catch (ServiceException ce) { throw new ProcessingException ("Could not obtain current time.", ce); } finally { manager.release(timeGiver); } }

And that concludes the tour. A parent service manager was initialized with a configuration obtained via JNDI and its components used by a Cocoon generator.

1. Comments

add your comments