DistributingCocoonApplications

RT from LeighDodds about how one might go about distributing a Cocoon application.

Scenario

Lets begin a thought experiment:

I'm building a web application based on the Cocoon architecture, which I'm intending to be given away as open source (of course!). I craft my application which ends up consisting of a number of Sitemap Pipelines, some XSLT stylesheets, assorted web pages and images, and perhaps some custom Components.

How do I package up this application so that others can make use of it? What options do I have?

Options

The available options seem to be:

1. **Complete Webapp** – a complete web application that is essentially a Cocoon distribution customised to run my application rather than the default set of examples.
2. **Donate to Cocoon** – I could donate my application directly to the Cocoon project. It could then be added to the standard Cocoon distribution, perhaps initially in the scratchpad. That way everyone who downloads Cocoon will have my application.
3. **Cocoon Bundle** – a zip/tar file consisting of just my application files, with instructions about how to configure an existing Cocoon instance to run my application, e.g. by UnderstandingCocoonMounts my sitemap.

What are the cost/benefits of each scenario?

The Complete Webapp option seems to be the best if I want people not already running Cocoon to run my application. The application distro is obviously much bigger, and existing Cocoon users may just want to add it to an existing instance.

Donating to Cocoon has the obvious benefit that a lot more people will see the application: it will be highly visible. It will already be pre-configured, so there's less involved in setting it up. However this makes the Cocoon distro itself much more complex, especially for people who aren't interested in my applications functionality.

Using a Cocoon Bundle has the advantage that its easy for existing Cocoon users to deploy. But has the disadvantage that non-Cocoon users won't be able to easily use it.

Conclusions and Suggestions

We might conclude from this that if I want to distribute a Cocoon application as widely as possible then I just simply bundle it as a Complete Webapp. But it's also worth my while to take the time to produce a Cocoon Bundle and supporting installation docs.

Applying this thinking to Cocoon itself it seems that one good way to modularise the application is to simply modularise the distribution along these lines. There could be a 'complete' distro that simply has everything – basically the current situation. However it would also be worth having an 'empty' distro that has nothing more than a MinimalSitemapConfiguration. There can then be separate Cocoon Bundles for additional self-contained functionality.

Another way to look at this idea is to think of Cocoon as a container, similar to a Servlet or EJB container: these application have a default install with little /no functionality, but often have separately downloadable examples. Application providers can bundle their applications for automatic deployment into a container.

Further Thoughts

Deploying a Cocoon Bundle could be an automated process. To mount the application I either need to:

1. copy the applications additional pipelines into the root sitemap, or
2. mount the application as a sub-sitemap

Both of these operations could be achieved with a simple XSLT stylesheet, possibly fronted by an Ant script. In the first case the stylesheet could just copy over the appropriate pipelines from one document to the other. In the second case, the stylesheet would just add the appropriate `<map:mount>` instructions #1.

Imagine that this application deployment functionality is a standard part of Cocoon#2. As an adminstrator I could use this functionality to automatically deploy (perhaps even download as well, cf: Debian package management) other peoples applications, those examples in which I'm interested, and even the documentation.

I've been mulling this over for a while, but the recent cocoon-dev discussions about modularising Cocoon (amongst other things) prompted me to actually write it down.

I'm in the process of writing a couple of Cocoon based applications, so wanted to get the options straight in my head. I prefer the Cocoon Bundle option myself

Comments very much appreciated.

– LeighDodds

See also

- BlocksUseCases how Blocks could help distributing applications

---

Readers comments

Automounting simple applications – BertrandDelacretaz

In this second case, if the application consists only of a sitemap and static files, copying them to a new directory under the mount subdirectory of a standard Cocoon installation is sufficient (see UnderstandingCocoonMounts).

Requirements for auto-installing Bundles – BertrandDelacretaz

Here's a tentative list of what Cocoon needs to be able to auto-install Bundle XYZ:

- Add entries to the main sitemap, marking them as created by Bundle XYZ so they can be removed automatically. Should be able to query the user for variable values.

- Same for cocoon.xconf (or add "sub-xconf files" owned by Bundle XYZ).
- Add jar files to the appropriate directory, marking them as owned by Bundle XYZ so they can be removed automatically.
- Restart Cocoon or otherwise reload the new jar files

Instead of modifying the existing files, it might be easier to create a new XYZ subdirectory under WEB-INF, and put all the Bundle configuration and jar files there. This requires changes to the sitemap and xconf handling though.

An interesting use-case for this is installing a Bundle that accesses a database using a new JDBC driver.This requires changes to sitemap, xconf and a new jar, and database information provided by the user during installation.

From a not-the-smartest-user-in-the-universe perspective:

If I recieve a "Cocoon-app" from somewhere, I want to:

1. drop it in a subdirectory / upload it via webdav to some directory I mean context.

Period. No restarts, no fiddling with the main site map, no configuration of the cocoon "kernel", only configuration of the new cocoon app.

Naturally one might have to do other configurations in the environment that the cocoon app might need, but that's a completely different story.

The important thing is that one should not have to risk to break anything in the existing cocoon installation, when installing new things.

/Ola Berg

What about "hot-deployment"? I think that the TreeProcessor and the ComponentManager used in Cocoon should be accesible and have hooks to an installer-thread. This thread could itself be a component that is bootstrapped by the CM. It listens for the bundle in the dropin subdirectory and automounts in .xmap and .xconf AND adds the component into the already running CM AND mounts the new sitemap components into the already running TreeProcessor (the last step shouldn´t be needed since the TreeProcessor checks for updates to the sitemap.)

/Mats Norén

The fact that cocoon basically is structured as an application server, similar to an EJB/WebApp container, is after a few days of rifling through the documentation, quite evident. Being used to J2EE packaging it has been quite confusing to figure out exactly **how** I am supposed to package my applications (together with cocoon) for distribution. When I say distribution here, I mean even within an enterprise. Sysadmins are an altogether separete breed from developers.

If cocoon is to be regarded as a (J2EE) contatiner in its own right alternative #3 should seriously be considered. But on the other hand, deploying cocoon as a webapp it seems quite strange to include a new webapp like structure (similar to a war file) inside the cocoon webapp. This does not really feel compatible with how webapps are handled in most servlet containers since entire applications are likely to be deleted when re-installing cocoon.

Probably a fourth alternative, similar to the way most APIs are handled, is the most natural: "support for cocoon as a library in an arbitrary webapp". All cocoon resources (including "system" stylesheets and so on) should be packaged nicely in jar files and maybe a sample web.xml file should be produced. Ideally the relevant jars should also be available as a libraries in a maven repository (like ibiblio) and cocoon dependencies should be clearly stated, perhaps even in a sample POM file.

Here's a related random thought: what about a "servlet archive", a .sar-file. This could be a jar-file with one or more servlet and all required resources similar to a war or (ejb) jar etc. Dependant libraries would be added to the manifest (MANIFEST.MF) as usual. The sar-file should include a default servlet descriptor for each servlet which could be referenced in the web.xml file where servlet file patterns should be set up. An analogy to the concept of a sar-file would be the JCA Adapters (rar-files), used to extend a J2EE-containers functionality to legacy systems. I do realize, of course, that this entails meddling with the J2EE-spec...

/Christoffer Soop