SimpleModProxy

Intro

ApacheModProxy is an excellent, in-depth article about why and how to use Apache HTTPD as a reverse-proxy front-end to Cocoon applications. That article is also rather *long*, so I'll summarize it here and encourage you to read it if you're interested in the long version.

So, by way of summary... you want to run Apache in front of your servlet container. The reasons:

- "Zero port 80 downtime" — the user *always* gets some page served, even if the application is down (rather than having their browser say, "could not connect");
- Much faster serving of static resources (e.g. images, CSS, client-side javascript, flash pieces etc.);
- Security — Apache can start as root, bind to (the privileged) port 80, and then switch to a non-privileged uid.

There's another compelling reason not discussed in ApacheModProxy, which is the ability to run multiple servlet engines. I'll discuss that later, but for now, let's get right to the good stuff!

The ApacheModProxy article gives several examples, starting from a simple set-up and working its way up to a gnarly, hairy one, just to show how elaborate it can be. You might be left with the impression that it really needs to be that involved. Beyond recapitulating the reasons for running Apache in front of Cocoon, the purpose of this article is to show how simple it can be.

A simpler way

A common feature of the examples in ApacheModProxy is that they all set DocumentRoot to the path of the webapp root directory. Then, some measures are taken to prevent the client from accessing the contents of WEB-INF/ (in accordance with the servlet spec). But that's really not all we want to do. We really don't want the sitemap, XML source documents, XSLT stylesheets, etc. accessible to the client, either. Fortunately, there's a really simple way to sequester all the static content and set things up so only the static content is directly visible to Apache.

My standard project layout includes this structure:

```
httpd/
    static           # symlink to... ----
webapp/                                 |
    static/       <----------------------
```

...and the DocumentRoot is set to *project_root*/httpd, not to the webapp directory.

All the static content is contained in `static`. That way,

- Apache can serve it up with one simple rule, instead of a bunch of RewriteRules matching filename suffixes, etc.;
- Cocoon can still serve up all the static content without Apache, so I can do development on my laptop where I do not want to have to screw around with Apache — I just point my browser at localhost:*port* and talk directly to the Cocoon instance (see "Serving Static Content" below).

So, all you might really need in your Apache configuration is

```
ProxyPass        /static/ !
ProxyPass        / http://localhost:8080
ProxyPassReverse / http://localhost:8080
```

`httpd` contains *nothing* except for the `static` symbolic link.

Multiple instances of Cocoon

Here's another reason run Apache in front of Cocoon: in production, it's a good idea to run each web application in a dedicated servlet container (Jetty is well-suited to this approach). Benefits:

- you can do things that require restarting the servlet container without affecting any other application
- you can crash the JVM without affecting any other application
- heavy load demands on one application don't (directly) affect applications running in other servlet container instances (i.e., on separate JVMs).

I use daemontools to start up (at boot time) and supervise all the Jetty+Cocoon instances (so that if one goes down, it's automatically restarted), and to log jetty's console output (using multilog).

Serving Static Content

As I mentioned, I want to be able to do development on my laptop without Apache, and that means the Cocoon app has to know how to serve static content by itself. Here's the sitemap snippet that does that:

```
    <!--
      ++  Static content
      -->

    <map:match pattern="static/**.css">
       <map:read src="{0}" mime-type="text/css"/>
    </map:match>
    <map:match pattern="static/**.js">
       <map:read src="{0}" mime-type="application/x-javascript"/>
    </map:match>
    <map:match pattern="static/**.gif">
       <map:read src="{0}" mime-type="image/gif"/>
    </map:match>
    <map:match pattern="static/**.jpg">
       <map:read src="{0}" mime-type="image/jpeg"/>
    </map:match>
    <map:match pattern="static/**.png">
       <map:read src="{0}" mime-type="image/png"/>
    </map:match>
    <map:match pattern="static/**.tiff">
       <map:read src="{0}" mime-type="image/tiff"/>
    </map:match>
    <map:match pattern="static/**.swf">
       <map:read src="{0}" mime-type="application/x-shockwave-flash"/>
    </map:match>
```

See SimpleContentModel for one approach to structuring the `static/` directory!

Author: MarkLundquist