Packaging Your Facelets in JARs

By Roger Keays, 17 July 2008

Because JSF automatically finds faces-config.xml in your jar files, it provides a neat opportunity for building a simple jar-based module system for your webapps. All you need is a way to bundle your templates and resources right? Well, thanks to Facelets it is actually very easy.

Facelets lets you implement your own ResourceResolver which is used to find templates. The following simple implementation extends the default resolver to check the classpath for the required template.

```
/**
 * This facelets resource resolver allows us to put facelet files in jars
 * on the classpath, as well as the context root of the webapp.
 *
 * @author roger
 */
public class TemplateResolver extends DefaultResourceResolver
        implements ResourceResolver {
    private Logger log = Logger.getLogger(getClass().getName());

    /** first check the context root, then the classpath */
    public URL resolveUrl(String path) {
        log.fine("Resolving URL " + path);
        URL url = super.resolveUrl(path);
        if (url == null) {

            /* classpath resources don't start with / */
            if (path.startsWith("/")) {
                path = path.substring(1);
            }
            url = Thread.currentThread().getContextClassLoader().
                    getResource(path);
        }
        return url;
    }
}
```

You enable this resolver by configuring your web.xml with:

```
<context-param>
  <param-name>facelets.RESOURCE_RESOLVER</param-name>
  <param-value>au.com.ninthavenue.webcore.application.TemplateResolver</param-v
</context-param>
```

That only gets us half way there, because we cannot use the ResourceResolver to serve external css, image and javascript resources. In order to do that, you can use a servlet which you will have to map one way or another in your web.xml. At Sunburnt, we don't actually map the servlet, but forward to it from a filter if a resource isn't found on the context path. The servlet looks like this:

```
/**
 * This servlet fetches a static resource from the classpath. Access to
 * java class files is restricted.
 */
public class ClasspathServlet extends HttpServlet {
    private Logger log = Logger.getLogger(getClass().getName());

    /** default constructor */
    public ClasspathServlet() {}

    /** serve the file from the classpath */
    @Override
    public void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException

        /* if this servlet is not mapped to a path, use the request URI */
        String path = request.getPathInfo();
        if (path == null) {
            path = request.getRequestURI().substring(
                request.getContextPath().length());
        }

        /* failure conditions */
        if (path.endsWith(".class")) {
            response.sendError(403, path + " denied");
            return;
        }

        /* find the resource */
        log.fine("Looking for " + path + " on the classpath");
        URL resource = Thread.currentThread().getContextClassLoader().
                getResource(path.substring(1));
        if (resource == null) {
            response.sendError(404, path + " not found on classpath");
        } else {

            /* check modification date */
            URLConnection connection = resource.openConnection();
            long lastModified = connection.getLastModified();
            long ifModifiedSince = request.getDateHeader("If-Modified-Since");
            if (ifModifiedSince != -1 && lastModified <= ifModifiedSince) {
                response.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
                return;
            }

            /* write to response */
            response.setContentType(getServletContext().getMimeType(path));
            OutputStream out = new BufferedOutputStream(
                    response.getOutputStream(), 512);
            InputStream in = new BufferedInputStream(
                    resource.openStream(), 512);
            try {
                int len;
                byte[] data = new byte[512];
                while ((len = in.read(data)) != -1) {
                    out.write(data, 0, len);
                }
            } finally {
                out.close();
                in.close();
                if (connection.getInputStream() != null) {
                    connection.getInputStream().close();
                }
            }
        }
    } /* doGet() */
}
```

Finally, you'll need to remember to build your jar with the resources included. We do this using maven by adding the following in the <build> section (after dependencies):

```
<build>
  <resources>
    <resource><directory>src/main/resources</directory></resource>
    <resource><directory>src/main/webapp</directory></resource>
  </resources>
</build>
```

That's the basic idea, I hope you followed along. There are a few things you should be aware of though:

- Relative URLs will be resolved from the same source. i.e. You can't use a relative URL inside a jar to resolve a template on the filesystem. That's only really a problem if you want your taglib in the jar but the actual tag files on the filesystem.
- This resolver may leak file handles if you are using facelets < 1.1.15 (due soon?). See Facelets Issue #278.

The JSR314 expert group have talked briefly about including something like this for JSF 2.0. I suspect it'll probably be possible in one way or another since it's the sort of thing you need for bundling Facelets-based components.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.