JSP in Sakai

In theory, JSP is supported natively by Tomcat. In pactice, if you try do this, Tomcat goes into an infite loop. Tomcat fields a JSP request and invokes the Sakai tool associated with it. If you use the Sakai RequestFilter, it detects the JSP page request and calls Tomcat to process it, thus creating a loop.

Fortunately, their are other ways to handle this. The approach describe below (in a series of experiments) uses JavaServer Faces to render JSP. JSP is (currently) a fully supported subset of JSF, so (in theory) JSP pages will be rendered and run. The experiments confirm this, with some limitations:

1. You need to bring in all of the Saka JSF dependencies in Maven.
2. Your web.xml file must be configured to use the FacesServlet correctly.
3. Your web.xml file needs to be configured to map files appropriately.
4. You need a minimal faces-config.xml.

It's not clear if you actually need a JSF Java model (a tool class). A minimal class with no data elements was included in the experiment al code.

JSP Experiments

The JSP Experiment - April 2006

Aaron Zeckowski and I tried a very simple experiment with an absolutely minimal JSP page. It seems that the Sakai request filter gets called by Tomcat before it processes the JSP request. The filter tries to forward it to Tomcat, causing a loop. We tried a bunch of changes how file mapper patterns were described, without success.

The JSF Hack - Part 1- May 27, 2006

It occurred to me today (May 27, 2006) that JSF fully supports JSP. I pulled my JSF books and confirmed this. I modified the JspTool experimental code that Aaron and I wrote in April. My thought was that by making the FacesServlet be the main controller, it render the JSP file (index.jsp). Well, you have to drag in eleven (!) maven dependencies to get JSF to compile and run properly. The web.xml file also has to be carefully constructed.

Alas, it doesn't work and I don't have the time at the moment to debug it. Likely a faces-config.xml file is needed. Perhaps a simpler approach would be to take an existing JSF example (such as the NoteTool) and strip it down to a JSP example. That's worth a try later.

The JSF Hack - Part 2 - May 28, 2006

Cloned the NoteTool example into JspTool. Deleted all pages except main. Renaming it to main.jsp failed, in spite of changing the mapping in web.xml. Once renamed back to main.jsf, displayed the Hello World HTML.

Added the line:

```
Date: <%= new java.util.Date() %>
```

Redeployed. Line is not interpreted and is displayed as is.

Added the line:

Date: <jsp:useBean id="dt" class="java.util.Date">

That didn't display either. Obviuosly, I'm missing something.

Adding the following line at the top didn't help either:

```
<%@ language="java" contentType="txt/html" %>
```

Made some changes to faces-config and web. Now get this error:

JasperException: /jsp/main.jsp(1,5) Invalid directive

Removing the JSP tags at least allowed the HTML to render. Now I need to get JSF to recognize and render JSP elements.

Ok, got it to work to some degree. The main.jsp file contains the following:

```
<html>
<body>
<h2>JSP Tool (.jsp)</h2>
Date: <% System.out.println("****** Test String ******"); %>
</body>
</html>
```

When run, this successfully prints the test string to catalina.out. This means that the <% %> tag is working, which I believe is the scripting tag in JSP.

Further exploration:

```
<html>
<body>
<h2>JSP Tool (.jsp)</h2>
<% System.out.println("****** Test String ******"); %>
<%! int ctr=10; %>
<%= ctr %>
</body>
</html>
```

This also runs, printing out "10" into the HTML document.

Removed the bundle and commented out reference to bundle in faces-config.xml. Simplified JspTool.java to bear minimum. It doesn't really do anything.

Conclusions

Properly configure, JSF can be used to render JSP pages. There may be further initialization needed to bring in JSTL and other tag libraries. Further experimentation is likely need to establish navigation, page transfer, bean access, etc.

Packaged up results in jsp2.zip.

Chris Davia

I have a Sakai Tool, written as a servlet.
The servlet has the usual attribute seeting:

```
request.setAttribute(Tool.NATIVE_URL, Tool.NATIVE_URL).
```

The servlet forwards to a jsp this way:

```
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(aURL);
dispatcher.forward(request, response);
```

The jsp creates some stuff, that generates some html. The HTML does a GET get back to the servlet. I am calling the servlet vai the servlet-mapping in the web.xml file.

So far so good, the doGet method in my servlet runs fine, but it seems to have lost its sakai session information. I get null pointer exceptions when getting the Placement from the ToolManager.

Am I supposed to call my servlet tool a different way, or is there something I need to do to pass the tool session information around?

Andrew Thornton

OK, Here's what I think you're doing:

- A browser performs a GET on /portal/site/siteId.
- This GET returns the webpage for the site identified by siteId.
- This webpage contains iframes, one of which contains the href for the tool placement of your tool. (/portal/tool/toolId)
- When the browser performs a GET on the above url this leads eventually to your Sakai Tool processing the request and eventually dispatching to the jsp.
- The jsp responds to the GET returning some HTML containing href urls (generated by the jsp.)
- The browser then requests one of the urls generated by the jsp.

So the question has to be; What is this url?

>From what I'm reading I take it is of the form /servlet-name/...? If so you're doomed. The url needs to be of the form /portal/tool/toolId/... If it isn't the portal does not handle the request and it does not get associated with a Sakai Session and it doesn't get the tool-placement id – or any of the other attrs – set in the request.

Glenn or others may know of a way to get Sakai to associate with such a request but AFAIK you've got to go through the portal.

You're not the only one to have been bitten by this.

andy

PS if you're using <c:url> you may want to rewrap the request and make getContext() return "/portal/tool/toolId", that'll make it <c:url> return the right thing.

Yes, request.getContext() before you set the Tool.NATIVE_URL attribute. That will return the "/portal/tool/toolId" bit.

I'm pretty sure there are other magical Sakai APIs that'll give you a proper urls but it depends on tightly you want to bind. I'm not sure that you necessarily want to give complete urls anyway though.

Glenn Golden

Please review the sample servlet tool. It has links back to "itself" (it is showing how a tool can create a URL space within itself for tool destinations) and a form post. The key code to form the URL for the post and the links is:

String destB = Web.returnUrl(req, "/b/123-45-6789");

The code to review is in the samples module, in ServletTool2.java. Web is in the Util module. The string in the code above is a tool specific path to include in the return URL. The majority of the URL is formed by the returnUrl() method, based on the current HTTP request. Note, if you have messed with the request to make it "native", unmess with it before using it for this sort of thing.

https://source.sakaiproject.org/svn/samples/trunk/sample-tool-servlet/ src/java/org/sakaiproject/sample/tool/ServletTool2.java

More Experiments - From Chuck Severance

I think that this is a very fruitful path. This is really cool.

I don't know the mechanics - but my instincts say that it would be nice to have a top level module in Sakai called "php" that includes necessary glue and some sample stuff PHP to allow people to do easy stuff in PHP.

Chuck Hedrick and Aaron Z are working no a "jsp" module that allows folks to do real work in JSPs.

Now neither PHP nor JSP are architecturally super-cool - but they are quite practical for bits and pieces and stuff that is already in PHP or JSP. And ease of use **is** important as we try to make Sakai appeal to a broader set of developers.

Might I suggest a contrib project called "php" where you accrete the reusable gems as you go?

Antranig Basman

Alistair Young wrote:

>>
>> Thanks Steve and Aaron,
>>
>> I think I see now, I think! No particular reason I was using a servlet
>> other than I thought that's what tools were. It seems however that
>> something internal is handling the requests:
>> uk.ac.cam.caret.sakai.WebappToolServlet
>>
>> There's a backing bean that spring loads up and into which it injects the
>> service and it goes into the spring application context:
>>
>> <bean id="itemsBean" ...
>> <property name="logic" ... (this is the "service"?)
>>
>> This is purely spring and nothing to do with sakai.
>>
>> In the jsp, to use the service you have to use the backing bean:
>> WebApplicationContext.getBean("itemsBeans");
>>
>> So there's a dependance on spring in all the jsps. Is the beauty of spring
>> such that you're meant to avoid dependance on the container?
>>
>> In the past I've used a listener to load up services and place them in the
>> context under known keys so servlets/jsps are kept clear of spring code.
>>
>> Sakai does it's tool stuff with the servlet-id.xml file.
>>
>
>>> > It uses the servlet id
>>> > to invoke your tool when someone navigates to it on a site
>
>> So the still unclear bit is what happens when you hit a tool in the sakai
>> ui. "something" causes output from the jsp. How is WebappToolServlet
>> involved in this? Is it some sort of request dispatching?

WebappToolServlet is a CARET invention (pace Andrew Thornton)
that was created to allow other Java view technologies than JSF
to work in the Sakai dispatching environment.
I'm **pretty sure** I remember you sitting in the talk where we
went over this but perhaps you were too busy trying to save the
nanoseconds
Without WebappToolServlet the context path part of the URL will
be mapped incorrectly, as well as various other parts of the
request object being empty. This correction cannot be made with
a simple filter, since on initial entry to the context there is
no valid URL.
Yes, Spring is meant to avoid all dependence on the container, but
this is plain JSPs here. This is no kind of supported option in
Sakai - you are basically "on your own" - although I'm
sure the few folks that have done JSP work in Sakai (Andy, Ian)
will be glad to help you out.
The supported/recommended option in Sakai until now has been JSF,
which is what the dispatching environment was designed for.
We are expecting a transition to RSF underway over these few
months.
The most direct step towards sanity you could take if you insist
on sticking with JSPs is to move to SpringMVC. This defines
various taglibs that should let you resolve beans as variables.
As a "framework" it's pretty useless however (naturally I would
say that...)

More from AZ - 2/23/11

I should add to this by mentioning that the JSP issues were solved and
I have used JSP + Spring MVC on a few recent projects. It works quite
well. I don't want anyone to get the wrong impression here. JSF is not
needed in order to use JSP in Sakai.

Here is a tool using JSP + Spring MVC. https://source.sakaiproject.org/svn/msub/unicon.net/kaltura/trunk/

Here is one that only uses JSP. https://source.sakaiproject.org/contrib/iclicker/trunk/