PortletSupport

NOTE: This is outdated information that applies only to Tapestry 3. For the current status of portlet support see

https://issues.apache.org/jira/browse/TAP5-64

Portlet Support

Many people ask for many things from Tapestry, but one thing that I've been hearing for well over a year now is a request to Portlet support. People want to use Tapestry to create Portlets. This could be a very good thing.

What's even better is that McKesson Provider Technologies has funded me (HLS) to provide portlet support for Tapestry 3.1.

Conversations about Tapestry will get somewhat more complex: there's *Servlet* Tapestry and *Portlet* Tapestry now!

Here's a thumbnail sketch:

- Each Portlet will be a separate Tapestry application, with its own HiveMind Registry
- Portlet action requests will be be handled like Servlet Tapestry ... but will not cause a (direct) render
- Portlet render requests will be mapped to the page service and a particular Tapestry page.
- A generic request/response/session/context facade will be placed around the Servlet or Portlet API objects
- Most services & components will deal with just the generic facade objects ... a limited number of services / components will have the real Servlet or Portlet objects injected.
- Certain services and pipelines will be duplicated, or other forms of indirection introduced, to pave over the differences between Servlet and Porlet operation

JSR-168 Notes

The first step has been to go over the JSR-168 Java Portlet Specification 1.0. Actually, I've configured and installed Jetspeed (2.0-M1) but haven't written even a basic portlet yet. What follows are my raw notes, which I'll amend as I learn things.

PLT.3 (page 16)

Persistent configuration and customization data sounds, to my ears, like another place to store persistent page properties.

The note about application-widescope appears, to me, to indicate ways that Portlets can communicate. Due to the complex class loader issues, I doubt that they can exchange complex data, however, and the opportunities for conflicts are rife. I understand that the more advanced portlet containers/engines include event mechanisms for portlet communication and coordination.

PLT.5.2.4 (page 24, line 21)

The render requests may be executed sequentially or in parallel without any guaranteed order.

That kills one of my initial ideas. There's a background issue involving generating unique names for Forms and for JavaScript elements (variables and function names). In Servlet Tapestry, the Body component is responsible for this (and a little bit of !IRequestCycle). In Portlet Tapestry, there's no way to coordinate across multiple Portlet renders (that are combined to form the page) ... especially given that some of the Portlet-generated content may be cached.

Later in the document, a *namespace* concept is introduced (for handling this specific case). That's good. Portlet Tapestry will need an equivalent to the Body component; or will need to provide similar behavior automatically. Part of Body's API will need to be generalized and abstracted into an interface ... possibly a HiveMind service.

PLT.5.2.4.2.1 (page 26)

Need to figure out the relationship between the request type (action request vs. render request) and the portlet mode. I kind of see an action request for, say, the help mode as linking to a "help" Tapestry engine service, which, in turn, will activate a Help page. This will cause render requests that will render the Help page.

Alternately, the portlet mode could be quite advisory as far as Tapestry is concerned. It may provide a page name when no specific page is provided (i.e., the view mode will activate the View page, the help mode will activate the Help page, etc.). Probably be a mapping somewhere between portlet modes and page names.

I believe that clicking the help control button (in the portlet decoration) will cause an action request with a mode of help. The render parameters will be set up for service=page and page=Help. Perhaps part of the rendered output is a PageLink; this will be a new action request (with service and page parameters, as with Servlet Tapestry). This will activate the second page and set render parameters to point to that new page.

There's no push to make Portlet Tapestry URLs pretty! The portlet URLs are already hideous and nobody cares ... and you don't tend to bookmark portlets (do you?).

DavidDeWolf:

- I'm not following the connection between the request type and portlet mode. I think they are much more independent than you may think.
- Typically a change in the PortletMode is used to alert the portlet that the user has requested alternative content. Because the portlet container maintains the state of a portlet when it switches modes, many portlets provide context sensitive help pages. To enable this type of functionality, a simple one to one mapping of modes to pages may not be enough.

HowardLewisShip: If the user clicks the "help" button, then the action request will be for portlet mode "help". In the absence of any additional parameters, Tapestry should activate the porlet's Help page. One you are on the help page, there may be additional links that activate other pages, say page "MoreHelp". The page service will create additional render parameters (service=page, page=MoreHelp) ... these will be available during subsequent render requests to get the MoreHelp page rendered.

I do see what you mean about context sensitive help (or editting). That's why I think there will be a service, "help", that will be invoked (in the absence of a service render parameter). The default implementation can activate the Help page, an application-specific implementaton could do something smarter. But you raise an interesting question ... I need to investigate what kind of information is available during a help action request (or any other mode for that matter) ... and I think the portlet will need to track an "active page" in the session.

PLT.5.2.4.4 (page 27)

May be some issues about propogating exceptions back to the Portlet API. Certainly PortletSecurityException needs to propogate back (I suspect the Portal will force a login page up). This may involve unwinding the exception stack, searching for portlet API exceptions.

PLT.7.1 (page 31)

I don't know that Tapestry will use render URLs. It's hard to say ... some engine services (such as page and external) are idempotent (as much as anything can be idempotent against a backdrop of changing authentication and server-side state).

I'm pretty sure that Portlet Tapestry will use action URLs for links and forms.

Thier "show customer summary" example could cut either way. To me, the Tapestry equivalent would be service=external, page=CustomerSummary, sp=Sfoo.com. The service parameter (sp) stores the customer id and is stored into a persistent page property. CustomerSummary gets activated, so a render URL properties are set: service=page, page=CustomerSummary.

DavidDeWolf:

- One thing to be aware of about action and request urls is that portlet specification does not require the container to maintain request attributes between the two request types. Because many containers implement the transition from an action request to a render request as a redirect, attributes bound to the ActionRequest will not be available for rendering. For clarity on this, see the Portlet1.0 Errata http://www.jcp.org/aboutJava/communityprocess/final/jsr168/Portlet_Specification_Errata.html#issue10

HowardLewisShip: I am aware of this, so my design is only going to rely on the action request setting render parameters, not attributes. These will be used for any subsequent renders before the next action request. Beyond that, we'll use portlet context attributes (equivalent to HttpSession attributes).

PLT.7.1.1 (page 32)

It seems like setting window state is *advice*. Services/components that need to know about the incoming window state, or need to change the window state for rendered links, will need to inject the ActionRequest and invoke setWindowState(). Likewise, ActionRequest.setPortletMode().

PLT.8 (page 35)

Normally, portlets, perform different tasks and create different content depending on the function they are currently performing.

I think with Portlet Tapestry, the portlet mode is less important (especially on render requests), since the service and page parameters will control what content is being displayed. The portlet mode (and window state) will be accessible to components that care. For example, pageValidate() may see if the window state is MAXIMIZED and may redirect to a different page that displays more detailed information (i.e., CustomerSummary for NORMAL and CustomerSummaryDetailed for MAXIMIZED). This may be something that can be controlled declaratively, using a HiveMind configuration. And, of course, there are customized window states.

PLT.11 (page 43)

The portlet-container must not propagate parameters received in an action request to subsequent render requests of the portlet.

This is good; the action request will have parameters for the requested operation (i.e., service=direct,page=View,component=link). At the end of the action request, well start with a clean slate, and load up render request parameters (service=page,page=View) that will be used for subsequent render requests until a new action request is processed.

If this isn't obvious yet: transient page properties on a Portlet Tapestry page will be of less use than in Servlet Tapestry. They'll be clear out at the end of the action request, and be null/default for the render ... even in the same request. This makes perfect sense given how portlets operate, but it means that where you might use a transient property in Servlet Tapestry you'll need to use a persistent property in Portlet Tapestry.

I was about to say ... *ideally, we'll have the technology in place to encode persistent properties as query parameters in 3.1.* However, that really isn't an issue ... whether it's PortletSession attributes or query parameters inside the render URL, it's still server side state. I wonder how portlet containers deal with browser back button and redirect-after-post issues?

DavidDeWolf:

- Because of the clear seperation between the action request and the render request, portlet-container's themselves typically use a redirect after post to transition from the action request to the render request.

HowardLewisShip: That's good to know. I need to write more experiments.

PLT.11.1.8 (page 46)

The portlet container is responsible for localization; this bypasses the standard Tapestry issue with a page changing locale, and then rendering in the old locale.

PLT.12.3.4 (page 53)

And here's the namespace needed to keep forms and JavaScript variables/functions from suffering name collisions.

PLT.14.3 (page 58)

Seems like a natural to "inject" preference attributes as properties, especially if they are read-only. Do we drive this via <meta> tags in the component spec, or do we introduce some additional spec elements? Can this be generalized, the way <inject> is?

PLT.15.3 (page 62)

Persistent page properties should be stored as PORTLET_SCOPE. If a portlet has information to share with other portlets, it should inject/obtain the Portlet Session and access its APIs directly.

Status

Feb 23 2005: Just getting started.