Portlet API 2.0 (JSR-286)

Ate Douma
JSR-286 Expert Group Member
Apache Portals Meetup, Amsterdam April 7 2008

Agenda

- JSR-286 status
- Limitations of JSR-168
- New features in JSR-286
- JSR-286 RI @ Pluto
- Pluto 2.0 roadmap

JSR-286 status

- http://www.jcp.org/en/jsr/detail?id=286
- Expert Group started: January 2006
- 1st Early Public Draft available: 2 August 2006
- 1st Public Review available: 24 July 2007
- Proposed Final Draft available: 13 December 2007
- Final Draft accepted: 3 March 2008
- Official release to be expected any day now (this week?)
- RI done under Apache Pluto umbrella with help from a developers group at University of Jena

Limitations of JSR-168

- inter-portlet communication / portlet coordination
- serving non-markup resources (.pdf, .doc, images etc.)
- portlet filters
- contributing javascript or CSS to <head>; using cookies
- proper support for common web frameworks
- AJAX support

Limitations of JSR-168

Inter-portlet communication

- only supported within the same portlet application using session attributes
- target portlets will only “see” messages during next render request
- portlets cannot (should not) update their state during a render request: “event” handling not really possible

Limitations of JSR-168

Serving non-markup resources

- A portlet can only render markup fragments
- Direct URL access to a portlet is not possible
- Requires direct servlet URLs for serving resource requests outside the markup fragment
- Might require (session) state coordination, like for security, between the portlet and the direct servlet requests

Limitations of JSR-168

Contributing to <head>, setting cookies

- Javascript or CSS can only be embedded withing the content markup; no body onLoad handling hooks
- No way to optimize or prevent duplicate contributions
- The API forbids adding cookies: only client side setting of cookies using javascript is possible

Limitations of JSR-168

Proper support for common web frameworks

- A lot of web frameworks are still primarily (or only) Servlet API oriented
- Servlet dispatching not supported from processAction
- Needs Portals Bridges or similar solutions
- Separate Action and Render request phases are difficult to incorporate or to migrate to
- “Unaware” servlets only “see” the application scope session
- JSP/JSTL support very limited:

```
<c:out value="<%= ((FooBean)renderRequest.getSession()
 .getAttribute("fooBean",PortletSession.PORTLET_SCOPE))
                                        .getBeanValue() %>"/>
```

Limitations of JSR-168

AJAX

- Portable solutions only using servlets are inefficient and ugly
- Many portals do provide native AJAX support now
- Native AJAX support with access to the portlet environment requires custom / portal specific extensions
  - But none of that is endorsed by the spec.

New features in JSR-286

- Java 5, alignment with J2EE 1.4 and WSRP 2.0
- Portlet coordination (Public Render Parameters, Portlet Events)
- Direct Portlet access using a ResourceURL with full control over the response
- Portlet Filters
- Extended Caching (private/shared, expiration/validation caching)
- Java 5 annotation support in GenericPortlet
- Access to the Portlet Window ID
- PortletURL generation callback listeners
- support for page <head> contributions and setting Cookies
- PORTLET_SCOPEd session attributes for dispatched Servlets
- Providing ActionRequest attributes to the RenderRequest
- Much improved support for common web frameworks

Public render Parameters

- Coordination of render state across web applications
- Limited to String values
- Defined in portlet.xml using Qnames:

```
<public-render-parameter>
  <identifier>foo</identifier>
  <qname xmlns:x="http://acme.com/parameters-2.0">x:foo.2</qname>
  <alias xmlns:y="http://acme.com/parameters-1.0">y:foo.2</qname>
</public-render-parameter>

<portlet>
  <portlet-name>PortletA</portlet-name>
  ...
  <supported-public-render-parameter>foo</supported-public-render-parameter>
</portlet>
```

Portlet Events

- Declared in portlet.xml using Qnames or a default namespace
- A portlet specifies which events it wants to receive or send

```
<default-namespace>http://acme.com/events-2.0</default-namespace>
<event-definition>
  <name>foo</name>
  <alias xmlns:y="http://acme.com/events-1.0/">y:foo</alias>
  <java-class>java.lang.String</java-class>
</event-definition>

<portlet>
  <portlet-name>Portlet-A</portlet-name>
  ...
  <supported-processing-event>
     <name>foo</name>
  </supported-processing-event>
  <supported-publishing-event>
     <name>foo</name>
  </supported-publishing-event>
</portlet>
```

Portlet Events (cont.)

- Wildcard support for matching to declared events:
  "`foo.event.`" matches both "`foo.event.one`" and "`foo.event.two`"
- The portal or portlet container can also send events
- New 3rd lifecyle phase: processed before rendering
- Can be send from both `processAction` and `processEvent`
- State changes are allowed during `processEvent`
- The portal / portlet container will act as broker
- Formally not 100% reliable, i.e. no guarantee of delivery (mainly for WSRP, "local" portals are less limited)
- API:

```
EventPortlet.processEvent(EventRequest req, EventResponse res)
StateAwareResponse.setEvent(QName name, Serializable value)
StateAwareResponse.setEvent(String name, Serializable value)
```

Resource Serving

- Invoking a Portlet directly (but still through the Portal)
- API:

```
MimeResponse.createResourceURL()
ResourceServingPortlet.serveResource(ResourceRequest, ResourceResponse)
```

- Extends the render phase, NOT a new life-cycle phase
- POST, PUT, DELETE supported
- Should not change shared portlet state
- Additional URL parameters are specific for the request
- Full control over request and response headers
- Can be used for binary data or “readonly” AJAX but cannot change navigational state

Portlet Filters

- Very similar to Servlet Filters both in definition and API
- Filters can be applied to multiple lifecycles
- Filters can be chained like Servlet Filters
- Wrapper classes for request and response objects
- Wildcards can be used for mapping to multiple portlets

```
<filter>
  <filter-name>Log Filter</filter-name>
  <filter-class>com.acme.LogFilter</filter-class>
  <lifecycle>ACTION_PHASE</lifecycle>
  <lifecycle>EVENT_PHASE</lifecycle>
</filter>

<filter-mapping>
  <filter-name>Log Filter</filter-name>
  <portlet-name>*</portlet-name>
</filter-mapping>
```

Extended Caching support

- Allow public cached content for multiple users

```
<portlet>
  ...
  <expiration-cache>300</expiration-cache>
  <cache-scope>public</cache-scope>
</portlet>
```

- Support validation based caching using Etag
- API:

```
response.getCacheControl()
  .getExpirationTime(), .setExpirationTime()
  .isPublicScope(), .setPublicScope(boolean)
  .getETag(), .setETag(String)
  .useCachedContent(), .setUseCachedContent(boolean)
```

Container runtime options

- Defineable in portlet.xml on both application and portlet level
- Queryable from PortletContext
- Four predefined options:
  - `javax.portlet.escapeXml` (for backwards compatibility)
  - `javax.portlet.renderHeaders`
  - `javax.portlet.servletDefaultSessionScope`
  - `javax.portlet.actionScopedRequestAttributes`

```
<container-runtime-option>
  <name>javax.portlet.servletDefaultSessionScope</name>
  <value>PORTLET_SCOPE</value>
</container-runtime-option>
```

renderHeaders

- Configured with Container runtime option
  `javax.portlet.renderHeaders` (default false)
- Override the `GenericPortlet.doHeaders` method for:
  - providing page <head> section element contributions, like CSS and Javascript <script> tags
  - other HTTP/Portal specific response header properties
  - setting Cookies
  - setting nextPossiblePortletMode
- API

```
org.w3c.dom.Element PortletResponse.createElement(String tagName)
PortletResponse.addProperty(String key, org.w3c.dom.Element element)
PortletResponse.addProperty(String key, String value)
MimeResponse.addProperty(javax.servlet.http.Cookie cookie)
RenderResponse.setNextPossiblePortletModes(Collection<PortletMode> modes)
```

renderHeaders (cont.)

- Contributing to <head> example:

```
protected void doHeaders(RenderRequest request, RenderResponse response) {
  org.w3c.dom.Element loadAjaxJS = response.createElement("script");
  loadAjaxJS.setAttribute("type", "text/javascript");
  loadAjaxJS.setAttribute("src", "wicket-ajax.js");
  response.addProperty(MimeResponse.MARKUP_HEAD_ELEMENT, loadAjaxJS);
}
```

(this is actually a bad example as Wicket will provide header contribution support natively and out of the box)

New features in JSR-286

Better web framework support

- Allow servlet dispatching, including using forwards, from:
  - `processAction`
  - `processEvent`
  - `render`
  - `serveResource`
- Optionally providing a `PORTLET_SCOPEd` session to Servlets
- Extended JSP tag library using new taglib uri
  `"http://java.sun.com/portlet_2_0"`
- Bridging and native integration underway for JSF (JSR-301) and Apache Wicket (1.3+)

New features in JSR-286

actionScopedRequestAttributes

- Action request attributes cached by the container and provided again during the render request
- Support transporting complex objects instead of Strings for which renderParameters should be used
- A required container feature!
- “Back” button supported through a `numberOfCachedScopes` cache size parameter.

```
<container-runtime-option>

    <name>javax.portlet.actionScopedRequestAttributes</name>
    <value>true</value>
    <name>numberOfCachedScopes</name>
    <value>10</value>
</container-runtime-option>
```

New features in JSR-286

~~Native AJAX support~~

- Was intended and discussed at large
- But: is not supported by this spec
- Difficulties:
  - AJAX request identification
  - synchronizing state changes to the client for other portlets on the page
  - XMLPortletRequest or plain XMLHttpRequest?
  - integration with popular AJAX toolkits
  - Too many not yet unknown or fully thought through use-cases
- *Let the community determine a feasible API first*

New features in JSR-286

... and even more ...

- Annotations: `@ProcessAction, @RenderMode, @ProcessEvent`

```
@ProcessEvent(qname="{http://acme.com/events}foo")
void processFoo(EventRequest request, EventResponse response)
  throws PortletException, java.io.IOException;
```

- `PortletRequest.getWindowID()`
- PortletURLGenerationListener

```
void .filterActionURL(PortletURL actionURL)
void .filterRenderURL(PortletURL renderURL)
void .filterResourceURL(PortletURL renderURL)
```

- "unmanaged" (private) custom Portlet modes

```
<custom-portlet-mode>
  <description>Creates content for Cut and Paste</description>
  <portlet-mode>clipboard</portlet-mode>
  <portal-managed>false</portal-managed>
  <decoration-name>ClipboardMode</decoration-name>
</custom-portlet-mode>
```

JSR-286 RI @ Pluto

- based upon a branch of Pluto 1.1.3
- Pluto 1.1.x development was moved to a branch too
- Pluto trunk development continued for version 1.2
- Early Jan 2008, JSR-286 branch was promoted to trunk (development for Pluto 1.2 was abandoned)
- February 7, 2008, JSR286-RI tagged from trunk, svn r619585, marking the final JSR-286 reference implementation code base
- Pluto 1.1.5 (from 1.1.x branch) released on March 17, 2008 (expected to be the last release from the Pluto 1.1.x tree)
- Pluto 2.0-SNAPSHOT development is still continuing!

Pluto 2.0 roadmap

- JSR-286 RI != Pluto 2.0
- The Pluto project is an independent Apache Project following its own requirements and standards
- Pluto Project: Pluto container + Portal Driver
- Features to resolve towards 2.0:
  - Improving upon the container SPI to allow better/full integration for Portals like Jetspeed-2, Cocoon Portal, etc., on the same level as was possible with Pluto 1.0.x (“fixing” Pluto 1.1.x simplifications)
  - But: striving to maintain Pluto 1.1.x easiness of use
  - Adding support/base implementations of some optional JSR-286 features (caching, custom modes/states etc.)
  - Code quality & performance improvements