Portlet 2.0 Preview JSR286

Cris J. Holdorph

JA-SIG Winter 2006

UNICON

1. Introduction
2. Portlet 1.0 History
3. Portlet 2.0 History / Future
4. Portlet 1.0 Feature Summary
5. Portlet 2.0 Features
6. Questions and Answers

Portlet 1.0 / JSR 168 History

- Java Community Process

  http://www.jcp.org/en/jsr/detail?id=168

- Led by Sun and IBM
- 1.0 Final Release Oct 27, 2003
- Interoperability between Portlets / Portals
- Set of APIs defining Portlets

Portlet 2.0 / JSR 286 History

- Java Community Process

  http://jcp.org/en/jsr/detail?id=286

- Led by IBM
  - Steven Hepper (sthepper@de.ibm.com)
- Early Draft Review Finished Sept 1, 2006
- Planned final draft by end of 2006
- Planned final release by May 2007
- Intermediate drafts available at
  - http://ipc658.inf-swt.uni-jena.de/spec

JSR 168 Feature Summary

- Lifecycle (init, **action, render**, destroy)
- Portlet URLs
- Portlet Mode (VIEW, EDIT, HELP)
- Portlet Window States (Normal, Maximize, Minimize)
- Render Parameters
- Portlet Preferences
- Portlet Session
- Portlet Deployment Descriptor (portlet.xml)
  - expiration-cache

JSR 286 Feature Summary

- Major Changes
  - Events
  - Shared Render Parameters
  - Resource Serving
  - Portlet Filters
  - Caching changes
- Minor Changes
  - Window ID
  - Namespacing
  - Request Dispatcher available to other phases
  - Portlet Taglib additions
  - Additional CSS classes

JSR 286 Feature Summary

- Unchanged
  - Portlet Modes
  - Window States
  - Portlet Preferences

Events and the Portlet Lifecycle

- New Lifecycle Phase – Event Processing
  - For each Portal Page request
    - Action – called on at most one portlet window
    - Event – called on as many portlet windows as necessary
    - Render – called on *up to* as many portlet windows that are displayed on current page
- Events may be generated during
  - Action phase
  - Event phase

Event Interface

- *javax.portlet.EventPortlet* interface
  - May be implemented by a Portlet
  - Contains one method
    - *void processEvent(EventRequest, EventResponse)*
  - *EventRequest* object provides event payload and other typical portlet values (mode, windowstate, etc)
  - *processEvent* is similar to *processAction* for copying renderParameters

- Events may be published using methods on *ActionResponse* or *EventResponse*
  - *setEvent*
  - *setEvents*
  - Multiple calls to *setEvent* and *setEvents* are allowed
  - Event processing order is not guaranteed

GenericPortlet support for Events

- *GenericPortlet* will implement the *EvenPortlet* interface
- *GenericPortlet*'s implementation of *processEvent* will dispatch to methods with appropriate runtime annotations

  @ProcessEvent(Retention=RUNTIME, name=<event name>)

- Typical implementations will be to subclass *GenericPortlet* and provide one method for each event type to be handled
- If no method is found *GenericPortlet* will just copy the render parameters from the *Request* to the *Response*

Events and portlet.xml

- Events must be defined in the portlet.xml
- After event definition, each portlet must declare what events it will publish or receive
- Portal defined events do not have to be defined in the portlet.xml
- Event names must be defined using the W3C QName standard
- Receiving event names are allowed to end with a * to indicate processing all events that start with the characters before the *

Events and JAXB

- JAXB 2.0 must be used to define the Event Payload
- JAXB is necessary for interoperability with WSRP events
- Implementing event payload class must be *Serializable* and annotated with JAXB annotations

Shared Render Parameters

- Shared Render Parameters may be visible to multiple Portlets
- Must be defined in the <portlet-application> of the portlet.xml
- Must be declared in each <portlet> section of the portlet.xml for a portlet that wants to receive the parameter
- Parameter name must follow the W3C QName specification
- A Portal is allowed to decide which shared render parameters will be shared by which portlets

Resource Serving

- Portlets can create two types of Resource Links
  - Direct Links
  - Resource URL Links
- Direct Links
  - More efficient
  - Not guaranteed to go through Portal
  - Will not have portal context available
  - Should only be used where access control is not needed
- Resource URL Links
  - Will go through the *ResourceServingPorlet* interface

ResourceServingPortlet

- *ResourceServingPortlet* interface contains one method
  - *void serveResource(ResourceRequest, ResourceResponse)*
- Portlet can produce content with
  - ResourceResponseWriter
  - OutputStream
  - Delegate with a RequestDispatcher call
- Portal is not allowed to modify content
- Portlet should not modify the portlet state
- May be used for *READ-ONLY* AJAX calls

Resource URL Links

- Resource URLs must not cause processAction to be invoked
- Portlet creates a ResourceURL to itself with *createResourceURL*
- A *ResourceURL* to a Portlet that does not implement the *ResourceServingPortlet* interface is an error
- Resource URLs cannot change the Portlet Mode or Window State
- Parameters on Resource URL are not render parameters

Portlet Filters

- Modeled after Servlet Filters
- Modify request data by wrapping request
- Modify response data by wrapping response
- Intercept invocation of a portlet after it’s called
- Filters may be chained

Portlet Filter

- Must implement *javax.portlet.Filter* interface
- Must provide a public no argument constructor
- *init()* method will be called on all Filters before being called on any Portlets
- *destroy()* will be called if Filter is removed from service
- *doFilter()* method called if *processAction(), processEvent(), render(),* or *shareResource()* would be called on Filtered Portlet

Portlet Filter

- Must be declared in portlet.xml in a <filter> tag
- A <filter-mapping> element must be declared to specify the portlets a Filter is applied to
- Order in portlet.xml matters for multiple filters of the same portlet
- Portal containers are expected/allowed to cache the “filter chain”
- A Filter may be restricted to specific lifecycle methods using the <lifecycle> element in the <filter-mapping> definition

Caching

- Two Types of Caching
- Expiration Caching
  - What existed before with some changes
- Validation Caching
  - New for Portlet 2.0 specification

Expiration Caching

- If no <expiration-cache> value is specified then portlet will be treated as always expired
- New <expiration-time> sub element
  - Previous *time in seconds* value goes here
- New <expiration-scope> sub element
  - PUBLIC_SCOPE may be shared across users
  - PRIVATE_SCOPE may NOT be shared across users
  - Default is PRIVATE_SCOPE
- Event dispatching will trigger expiration cache the same way Action dispatching does
- <expiration-time> and <expiration-scope> may be set programmatically at runtime

Validation Caching

- Portlet should set an ETAG property and expiration-cache time when writing render output
- New render requests will only be called after expiration-cache time is reached
- Render request will be sent ETAG
- Portlet should examine ETAG and determine if cache is still good
  - If cache is good, set a new expiry time and NOT render any output
- Portlet must set the ETAG, expiry time and caching scope before writing any output

Window ID

- New *PortletRequest.getWindowID()* method must return the Portlet Window ID
- Review from JSR 168
  - "Portlet Deployment" (not mentioned directly in specification)
    - portlet.xml file information
  - Portlet Definition
    - Publish time information
  - Portlet Entity
    - Subscribe time information
  - Portlet Window
    - Login / Session time information

Namespacing

- New *PortletResponse.getNamespace()* method must provide a unique value for the current Portlet Window
- *getNamespace()* must return the same value for the lifetime of the Portlet Window
- Value may be used to prefix Javascript functions / variables or other items within a portal page that must be unique

Request Dispatcher

- Request Dispatcher may now be called from *processAction()* and the new *processEvent* method
- All non-render lifecycle methods will not be allowed to be write to any output stream
- Portlet Request Dispatchers must follow any Servlet Filters set up

Portlet Tag Library

- New *resourceURL* tag
- Existing *namespace* tag must match the value of *PortletResponse.getNamespace()*
- *copyCurrentRenderParameters* attribute on Action and Render URLs
  - A boolean (default false) setting indicating whether all current render parameters should be made parameters of this URL
- *escapeXML* attribute on Action, Render and Resource URLs
  - A boolean (default true) specifying whether characters in output should be converted to entity codes

Additional CSS Classes

- Additional CSS Classes are defined in the new document
- Some additional classes currently share names with existing classes
- See Appendix C

What's Missing?

- Ability to change the state of a portlet with AJAX calls
  - Forthcoming in JSR 286
- Shared Session Attributes
  - Removed from JSR 286
- More CSS classes
- Better Groups and Permissions support

Questions?

Cris J. Holdorph

holdorph@unicon.net

www.unicon.net