Portlet Specification 3.0 Is Here!

[CON3860]

Simplifying Portlet Development

Martin Scott Nicklous
JSR 362 Specification Lead & Software Architect,
IBM Deutschland Research & Development GmbH

Neil Griffin
Software Architect,
Liferay Inc

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

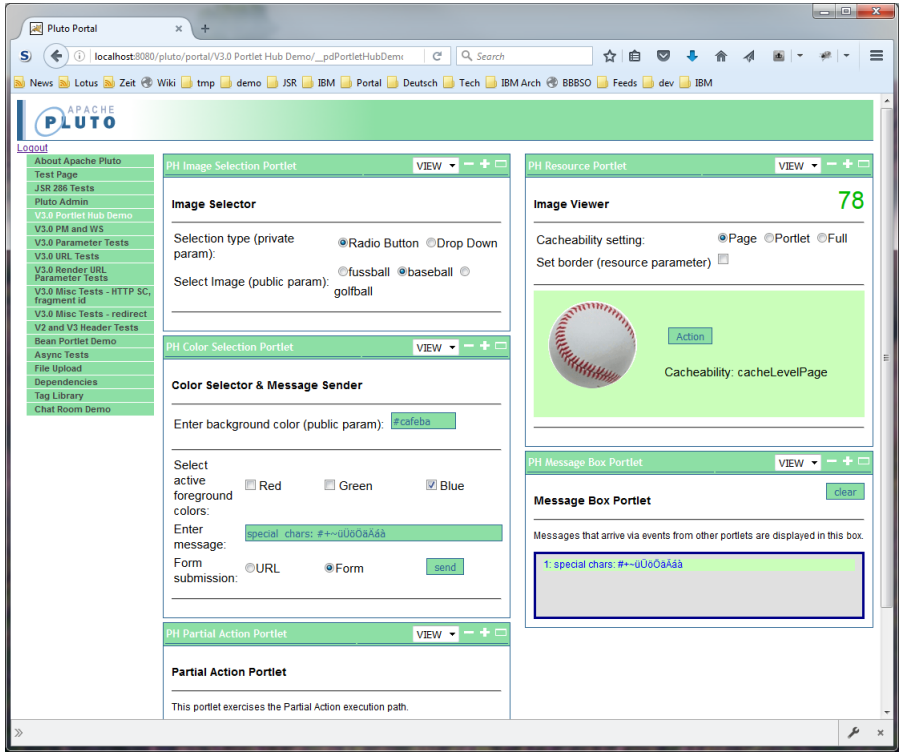
IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

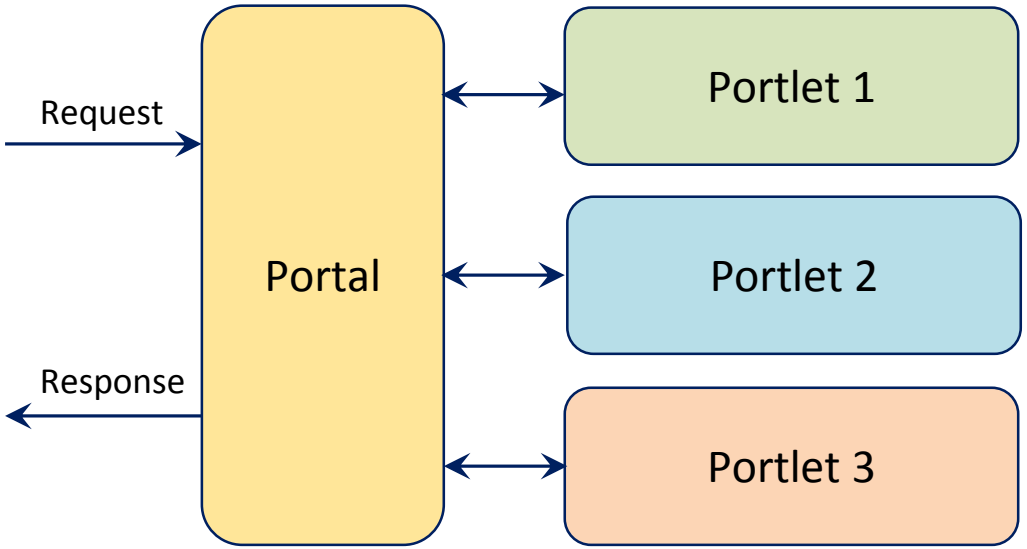
IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

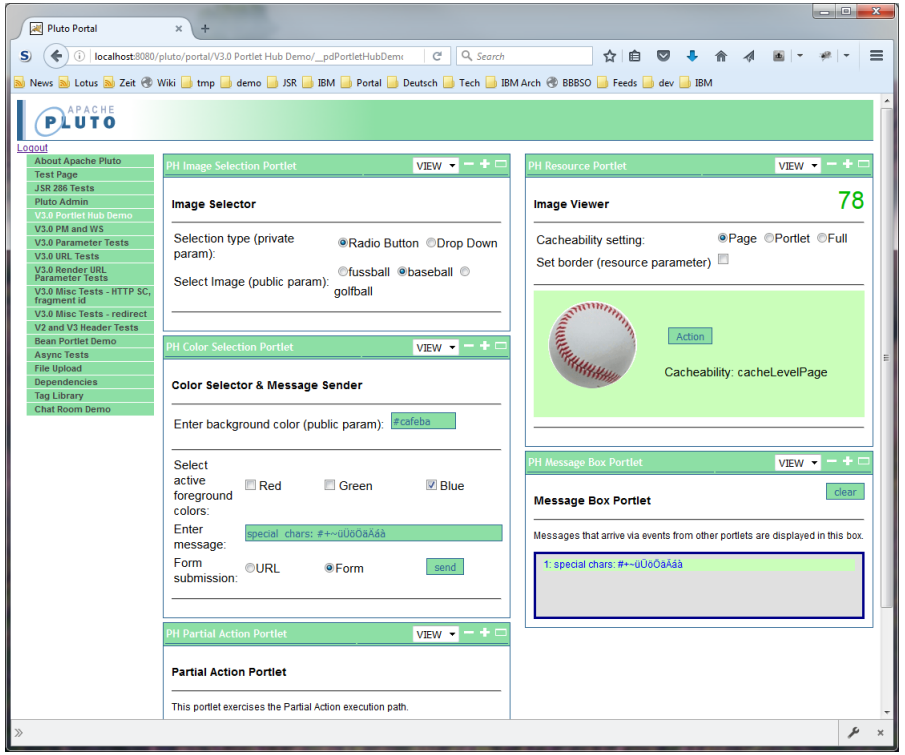
What is a Portlet?

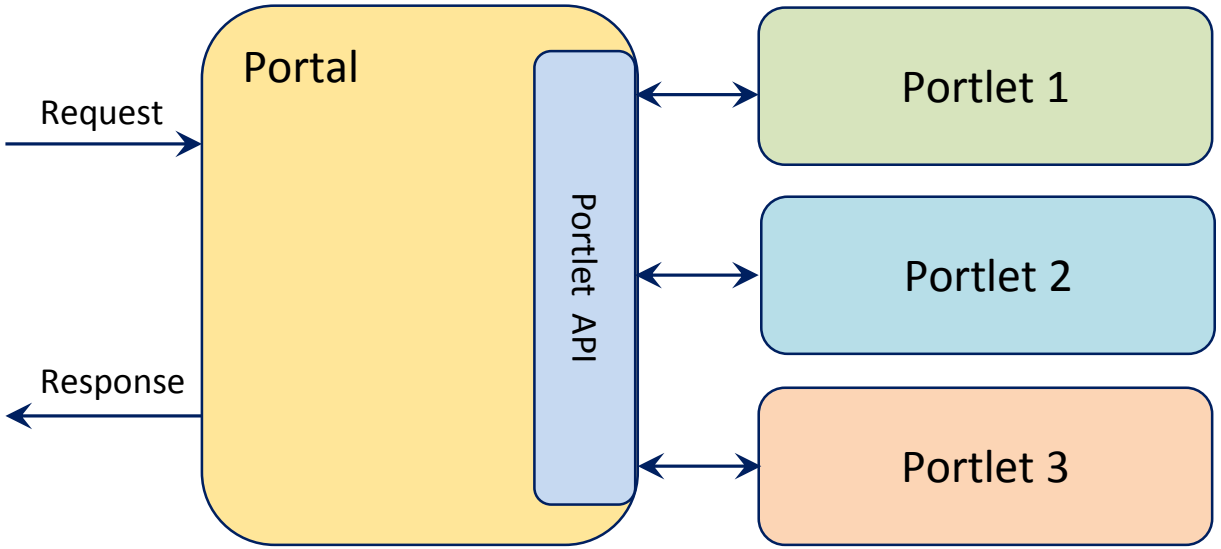




- A java-based web application that produces markup fragments for integration into a web page
- Each portlet is an independent application
- Uses a standard Portlet API defined by JSR 362 Portlet Specification

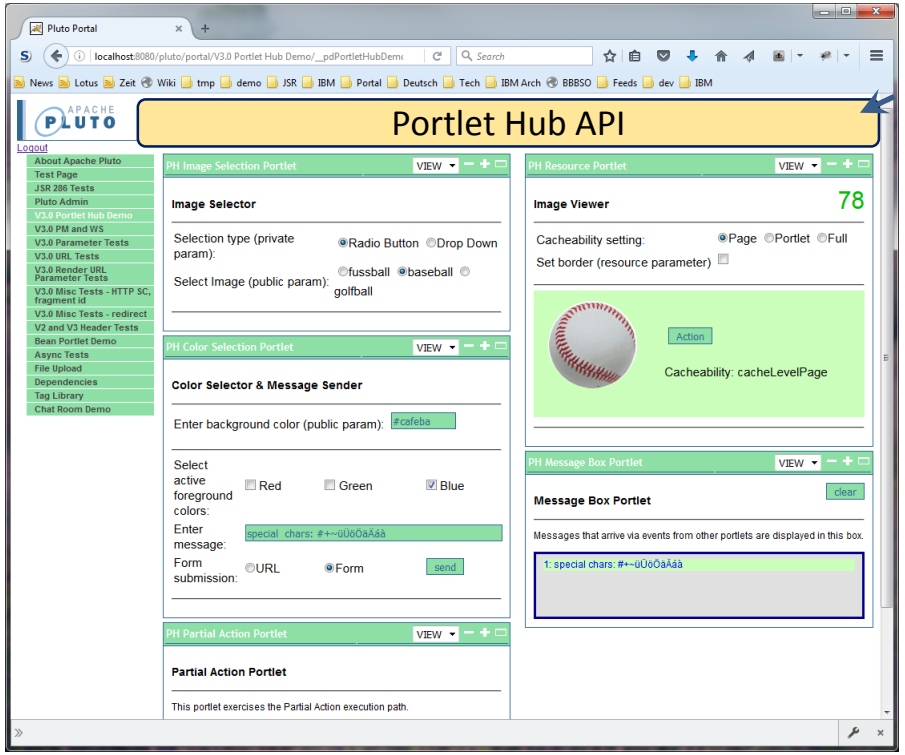
The Portlet API

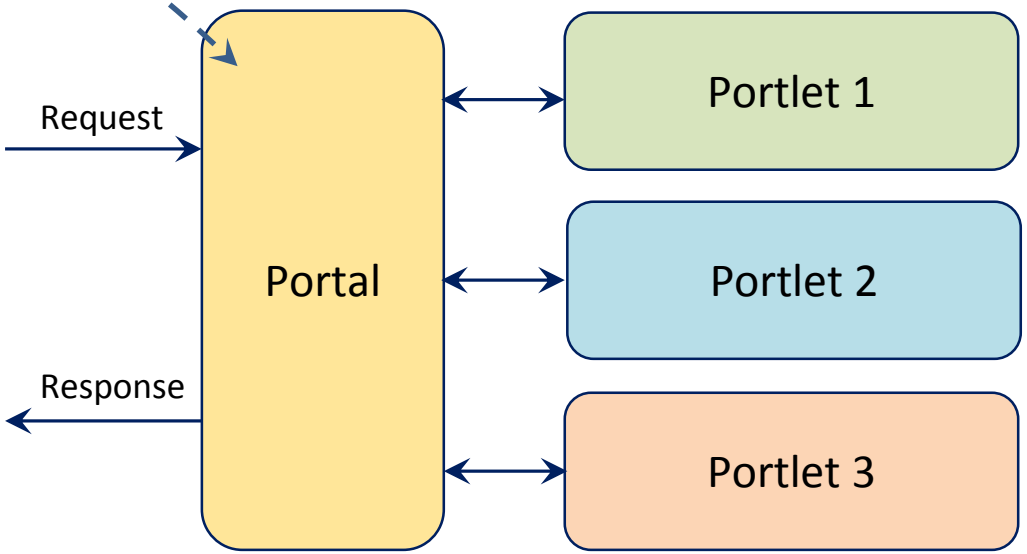




- Portlet container calls the portlet lifecycle methods
- Provides services to portlet such as URL generation, portlet preferences
- Coordination between portlets: Public render parameters, events

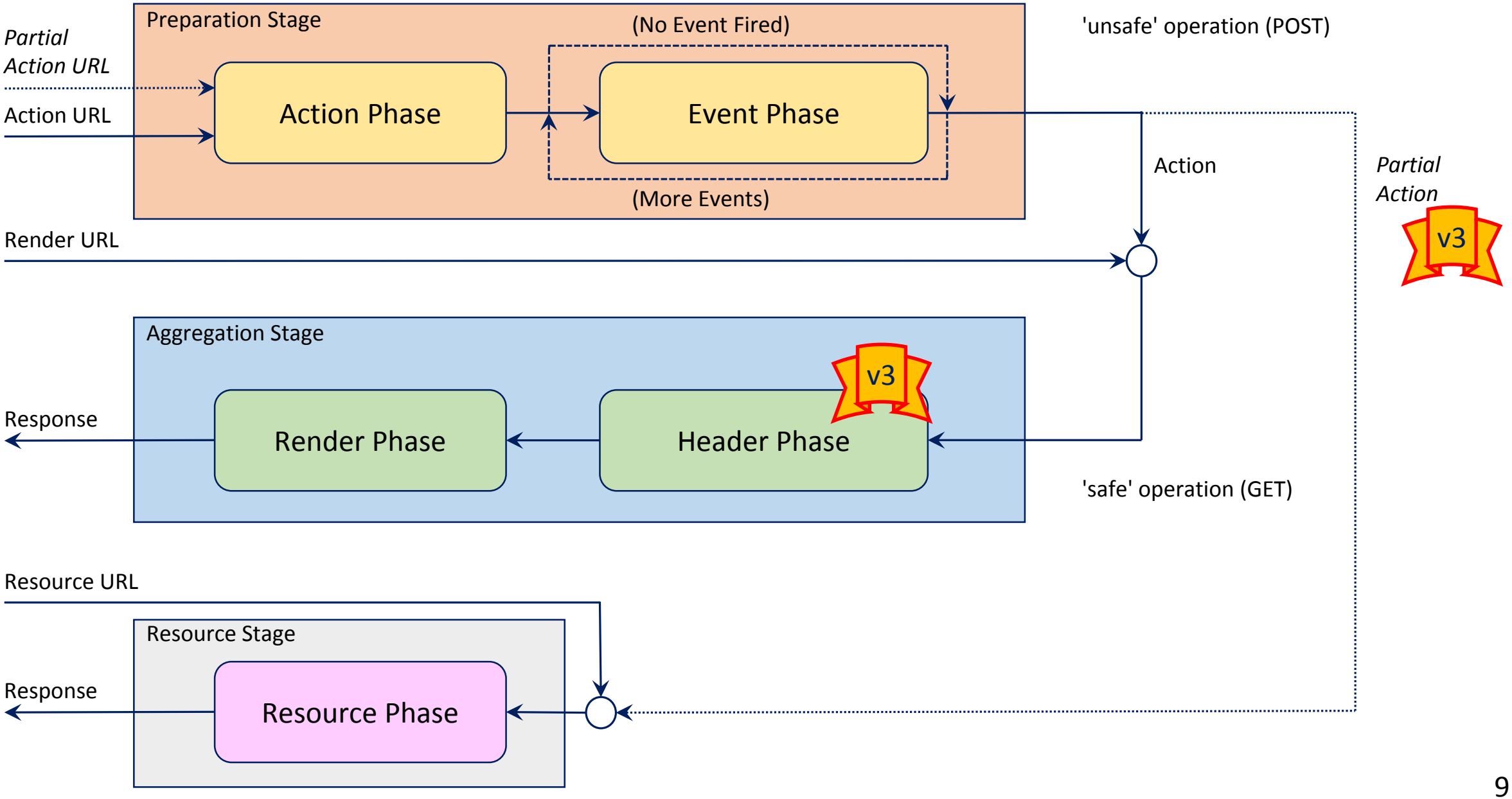
The Portlet Hub





- Standardized by JSR 362 Portlet Specification 3.0
- JavaScript module running on the browser
- Provides services to portlet JavaScript code (the 'portlet client')

The Portlet Phase Model



The Portlet Lifecycle Methods

Preparation Stage

Action Phase
Action Method
processAction(ActionRequest, ActionResponse)

Event Phase
Event Method
processEvent(EventRequest, EventResponse)

Aggregation Stage

Render Phase
Render Method
render(RenderRequest, RenderResponse)

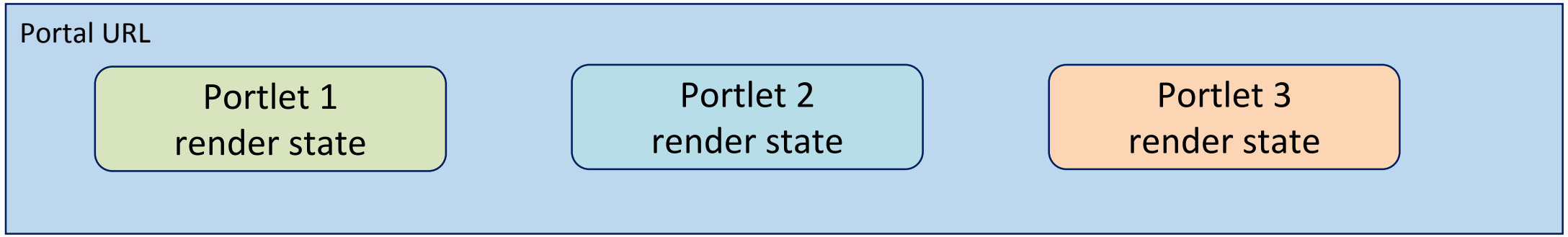
Header Phase
Header Method
renderHeaders(HeaderRequest, HeaderResponse)

v3

Resource Stage

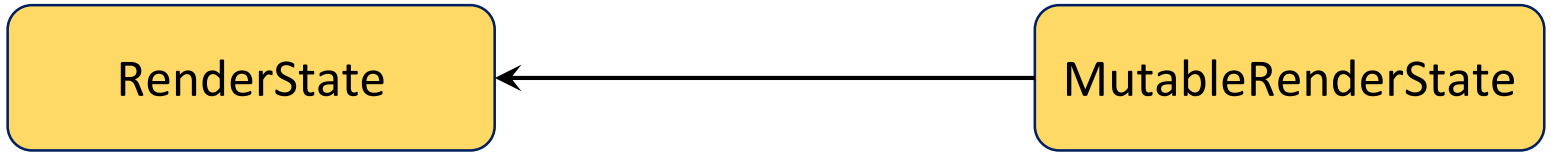
Resource Phase
Serve Resource Method
serveResource(ResourceRequest, ResourceResponse)

Central Concept: The Render State



- Managed by the portal for all portlets on the page
  - Render Parameters
    - String name / value pairs; the value is a String array
    - Each render parameter can be either private or shared with other portlets
  - Portlet Mode
    - special parameter indicating type of display ... Regular view, help screen, etc.
  - Window State
    - special parameter indicating how much is displayed ... Full screen, minimized, etc.
- Stored (conceptually) in the portal URL

The Render State Interfaces

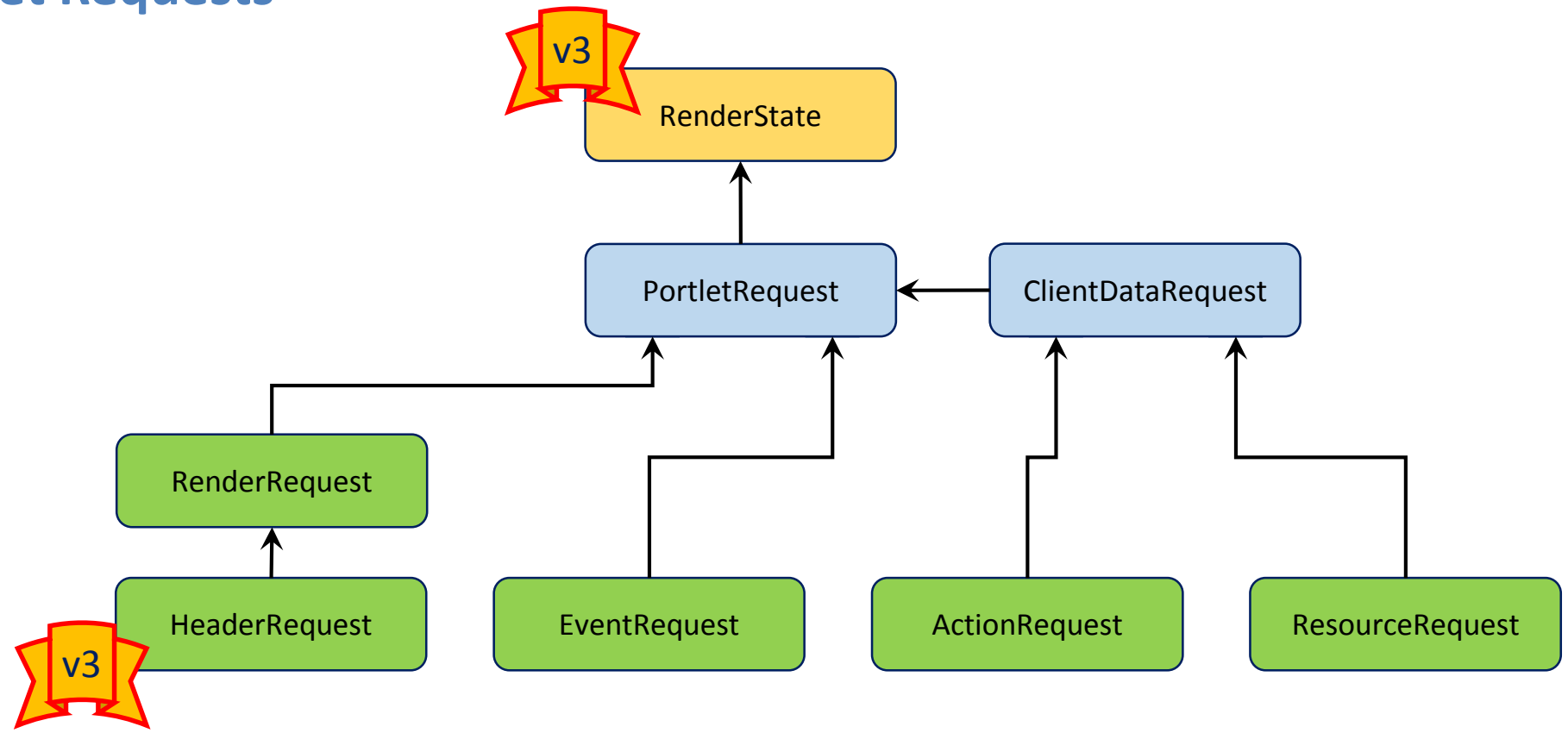


- Getter methods
- Setter methods

- New: Render State methods factored out into separate interfaces
  - Common interfaces for portlet requests, responses, URLs

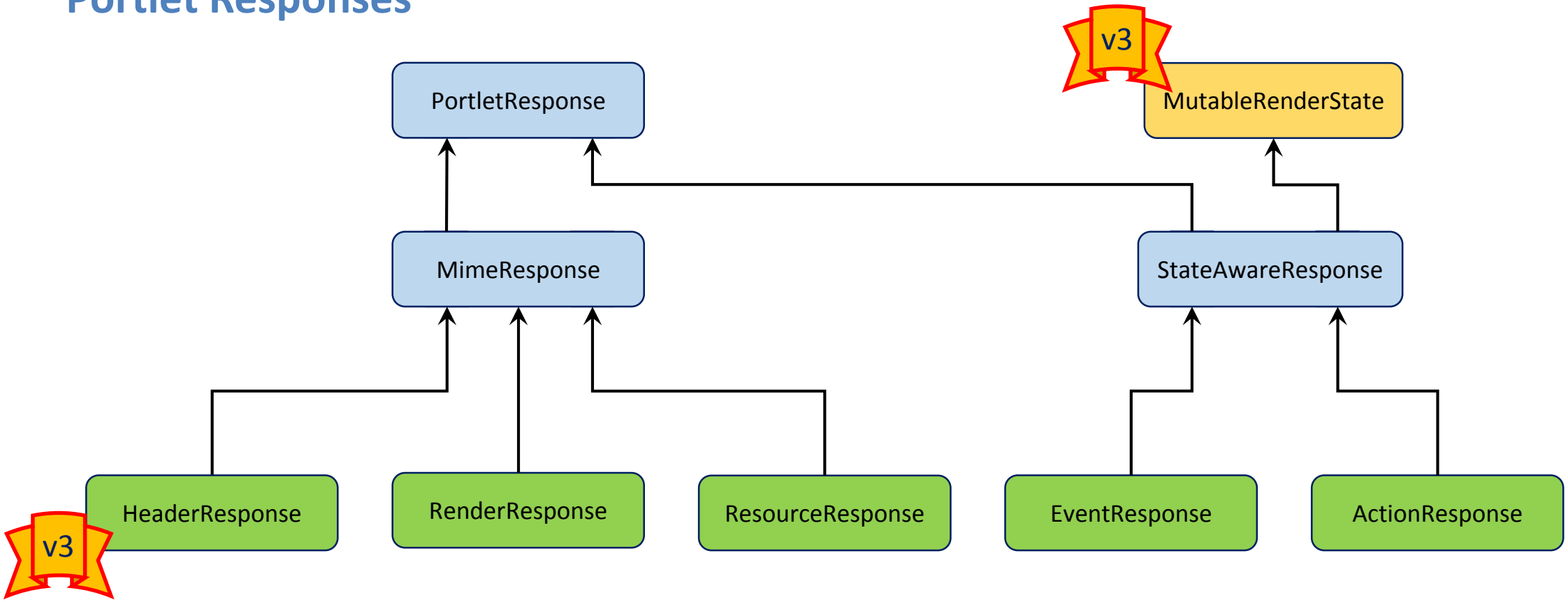
- Allow portlet to read the render state during all processing phases
- Allow portlet to set the render state in preparation stage and on URLs

Portlet Requests



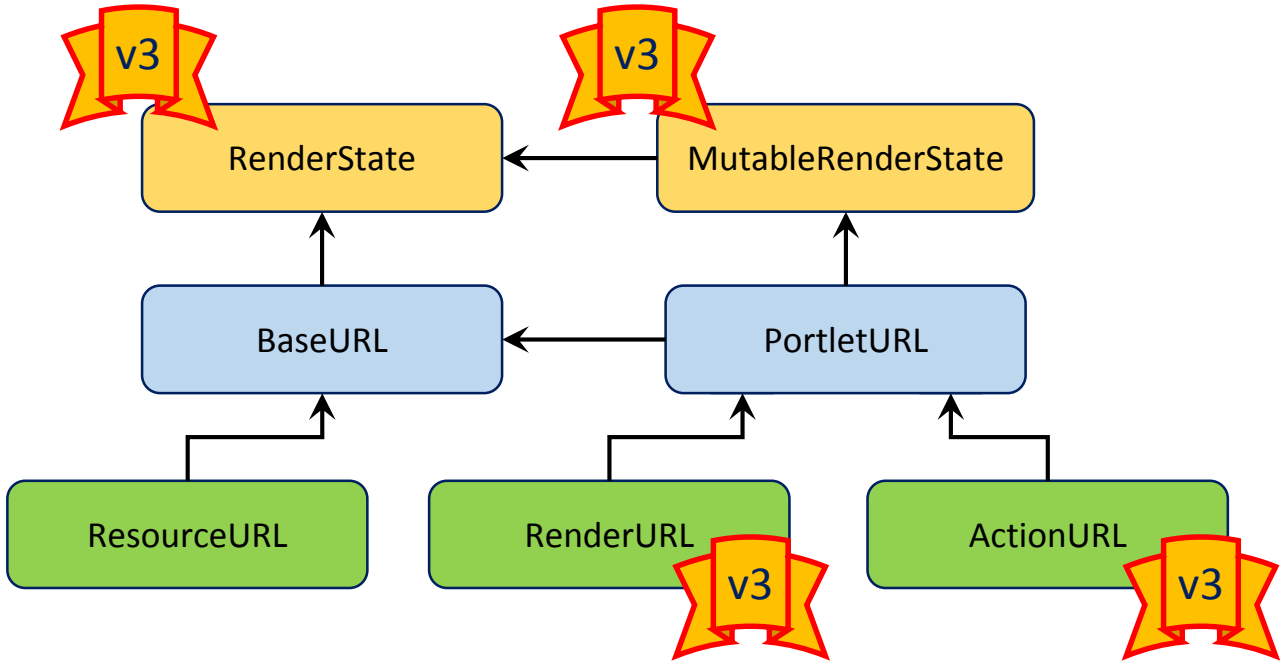
- Request interfaces extend render state interface for read-only access
- Added HeaderRequest interface

Portlet Responses



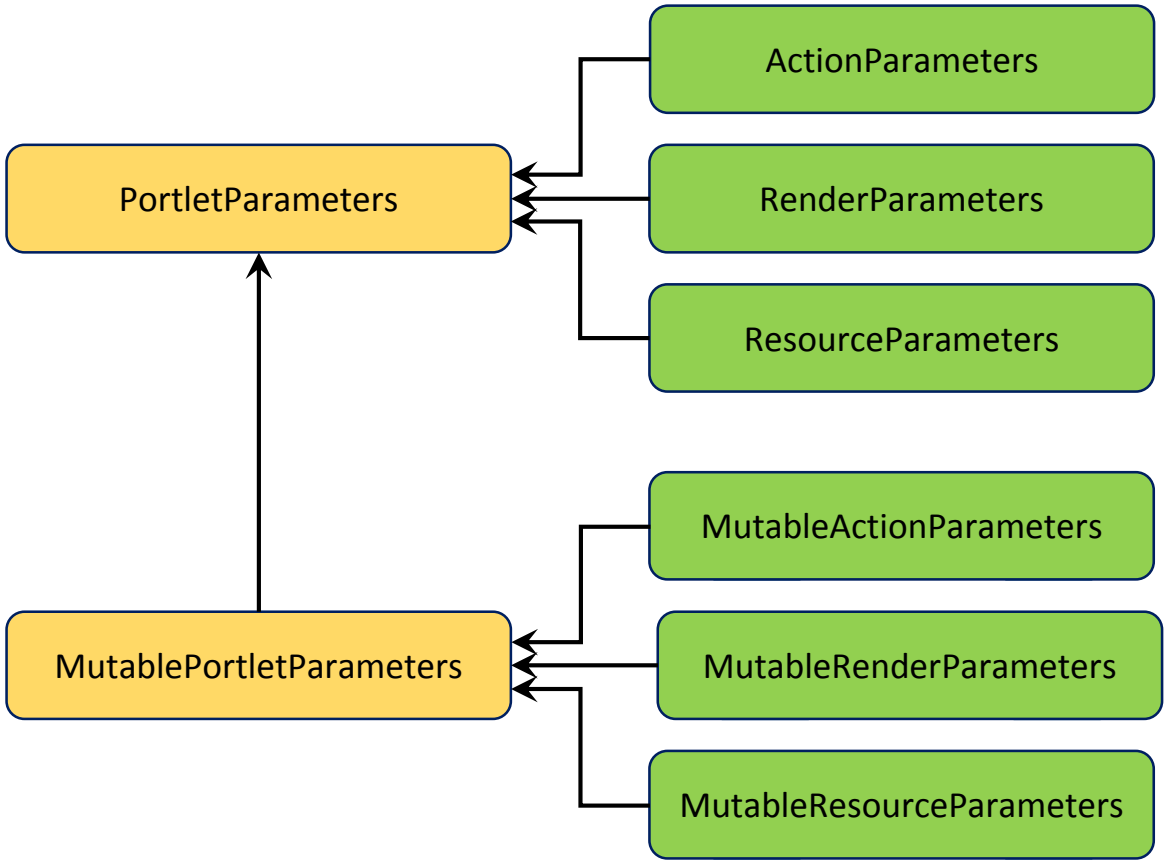
- Response interfaces extend mutable render state interface
- Added HeaderResponse interface

Portlet URLs



- URLs can be created during render and resource phases
- URL interfaces extend common render state interfaces
- Added dedicated interfaces for render URL and action URL

Portlet Parameters



- Dedicated objects for each parameter type
- Render parameters
- Action Parameters
- Resource Parameters

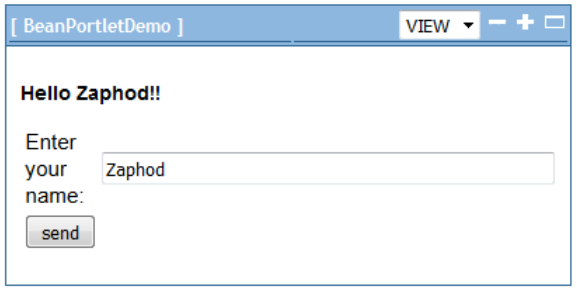
CDI Support

- Portlet classes are instantiated through the CDI container
  - To enable dependency injection
  - Scope annotations on the portlet classes are respected
- Dependency injection supported in portlet filters & listeners
- Many portlet artifacts made injectable
  - Request & response objects, parameter objects, portlet config, ...
- Custom CDI scopes for portlets
  - The portlet session scope - @PortletSessionScoped
  - The portlet request scope - @PortletRequestScoped
  - The render state scope - @RenderStateScoped

Extended Method Annotations

- Portlet can implement lifecycle methods by:
  - Implementing the Portlet interface
  - Extending the GenericPortlet class
  - Annotating a method within a valid bean class
- Using the extended method annotations
  - Portlet methods can be in different bean classes
- The extended method annotation
  - Must specify the portlet name
  - In some cases, multiple names or a wildcard '*' can be specified
  - Can contain additional info depending on the method

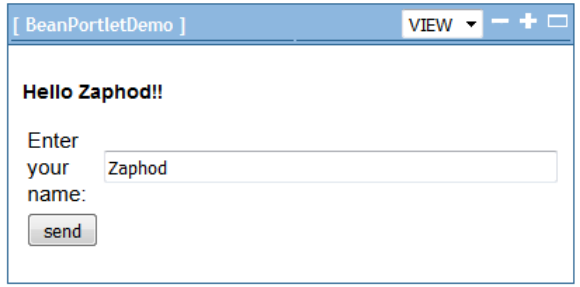
@RenderMethod




```
@RenderMethod(portletNames = "BeanPortletDemo")
public String simpleRender() {
    StringBuilder txt = new StringBuilder(128);
    ...
    return txt.toString();
}
```

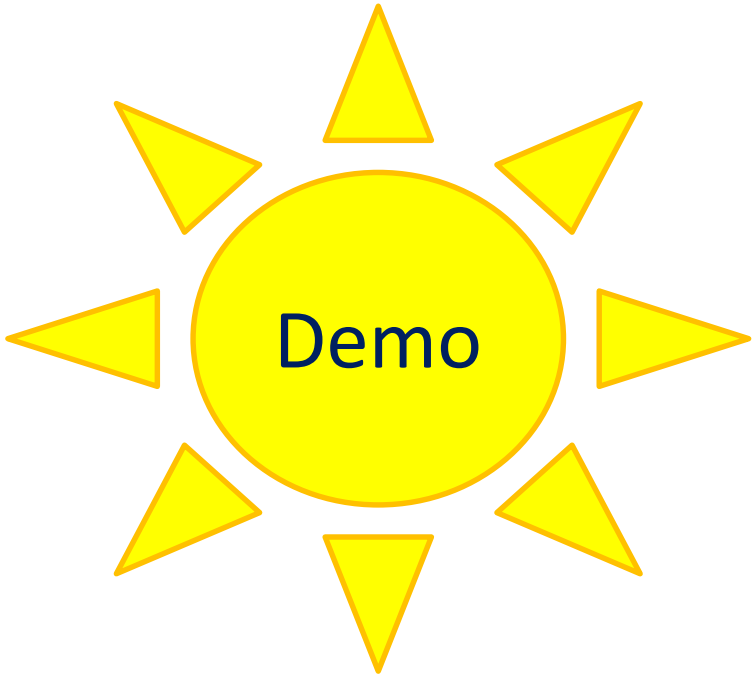
- Produces markup during render phase
- Portlet mode can be specified in annotation
- Resource (servlet, JSP) can be included after method execution
- Multiple render methods for single render phase execution allowed
- Simplified method signatures

@ActionMethod



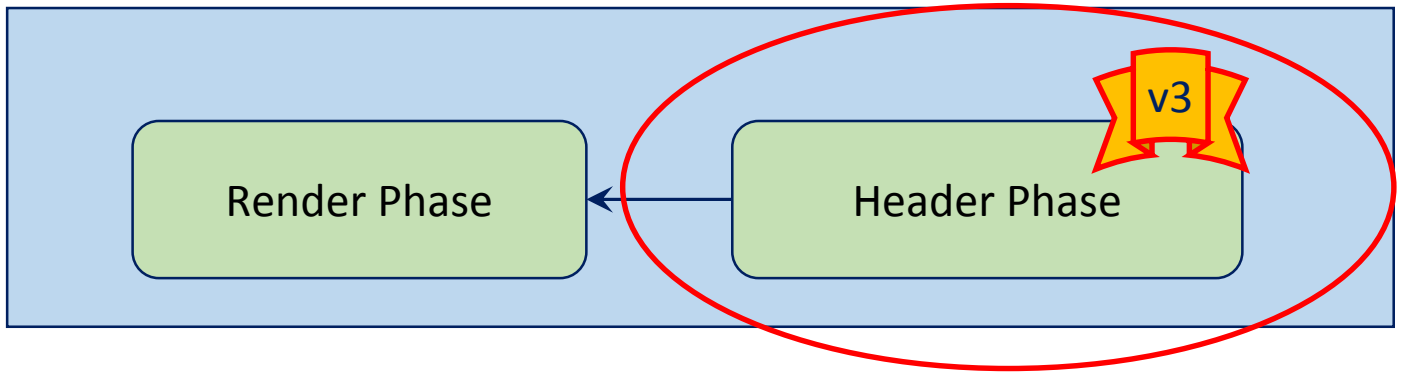

```
@ActionMethod(portletNames = "BeanPortletDemo")
public void setName(ActionRequest request,
                    ActionResponse response) {
   ActionParameters aparams = request.getActionParameters();
   MutableRenderParameters rparams = response.getRenderParameters();
   String name = aparams.getValue("name");
   renderParameters.setValue("name", name);
}
```

- Processes forms during action phase
- Form parameters available in method as action parameters
- Action name can be specified
- Event references for publishing events can be specified
- Simplified method signatures



- Begin with empty class; add hello world render method
  - Simple portlet configuration
  - Using the @RenderMethod annotation implicitly defines a portlet
  - The portletNames attribute defines the portlet name, but can also define a list of names
  - The portlet shown does not implement any portlet API specific interface
  - Shows use of simplified method signature - returns a string
- Add the NameBean.java class
  - An @RenderStateScoped bean
  - When the bean is passivated, the bean state is stored as a portlet render parameter
  - The parameter name can optionally be specified as annotation attribute
- The bean must implement the @PortletSerializable interface
  - To store the bean state as render parameter values String array
  - Allows the bean state to be a public render parameter when param name specified
  - Allows the bean state to be accessed on client when param name specified
- Shows injection of portlet String namespace. Same value returned by PortletResponse#getNamespace()
  - Uses @Namespace qualifier annotation
- Shows injection of portlet MimeResponse object
- Adds a second @RenderMethod with ordinal number = 200
  - When a portlet has multiple render methods for a given portlet mode, they are rendered in ascending order
  - Supports a portlet component model
  - Shows that portlet lifecycle methods do not need to be located in the same class
  - Shows use of V3 ActionURL interface
  - ActionURL is created using the injected MimeResponse object
  - Use of Copy.ALL to copy all current render parameters to new URL
- Injects a portlet ActionParameters object
  - Can only be accessed from within an action method
- The @ActionMethod defines an action method with actionName attribute 'setName'
  - Method name can be freely selected
  - Relaxed method signature requirements - don't need to declare throws clause if not needed
  - Action method is executed when a form targeting the portlet is submitted
- Look at Help.java
  - Includes a @PortletConfiguration annotation that defines support for the 'help' portlet mode
  - Includes a render method that includes the help JSP
  - The portletMode="help" attribute specifies that this method is to be executed to display help

The Header Phase

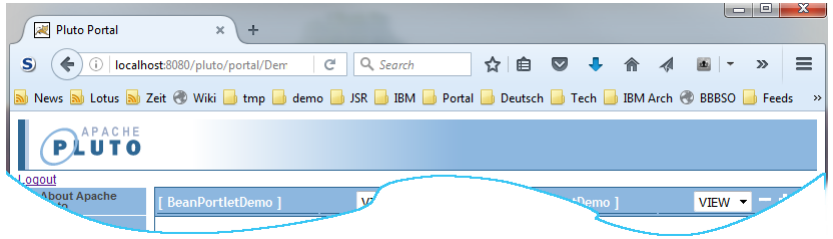


- Header phase is executed before the overall portal response is committed
- Markup written to output is added to document HEAD section
- HTTP headers can be added
- Portlet can declare dependency on page resources
  - Portal is responsible for placing resource on page

@HeaderMethod

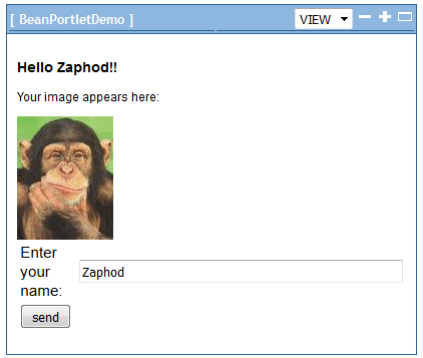
v3

```
@HeaderMethod(portletNames="*")
public void header(HeaderRequest req, HeaderResponse resp) {
    resp.addDependency("PortletHub", "javax.portlet", "3.0.0");
}
```



- Produces HTTP headers and <HEAD> section markup during header phase
- Portlet mode can be specified in annotation
- Resource (servlet, JSP) can be included after method execution
- Multiple header methods for single header phase execution allowed
- Simplified method signatures

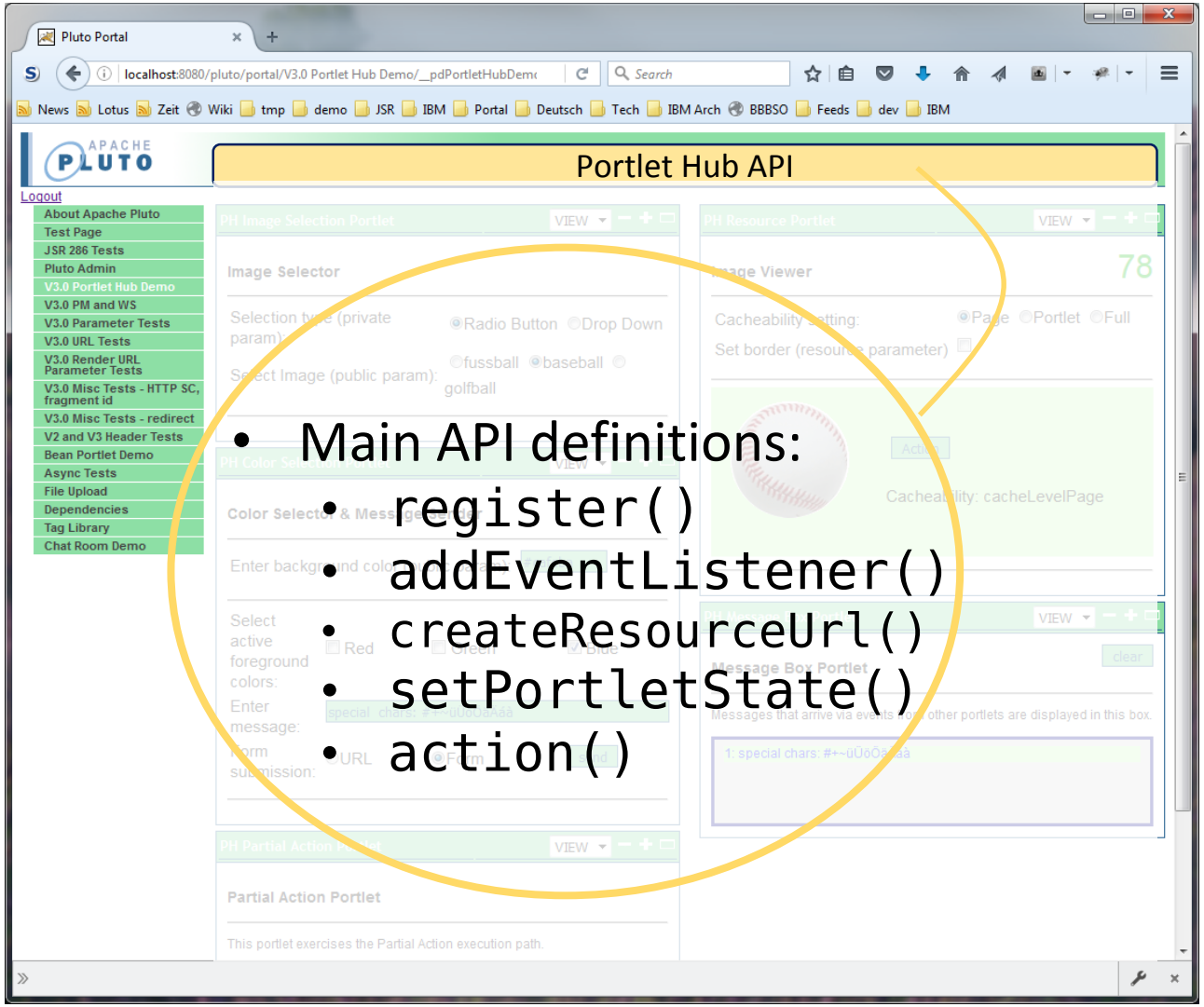
@ResourceMethod



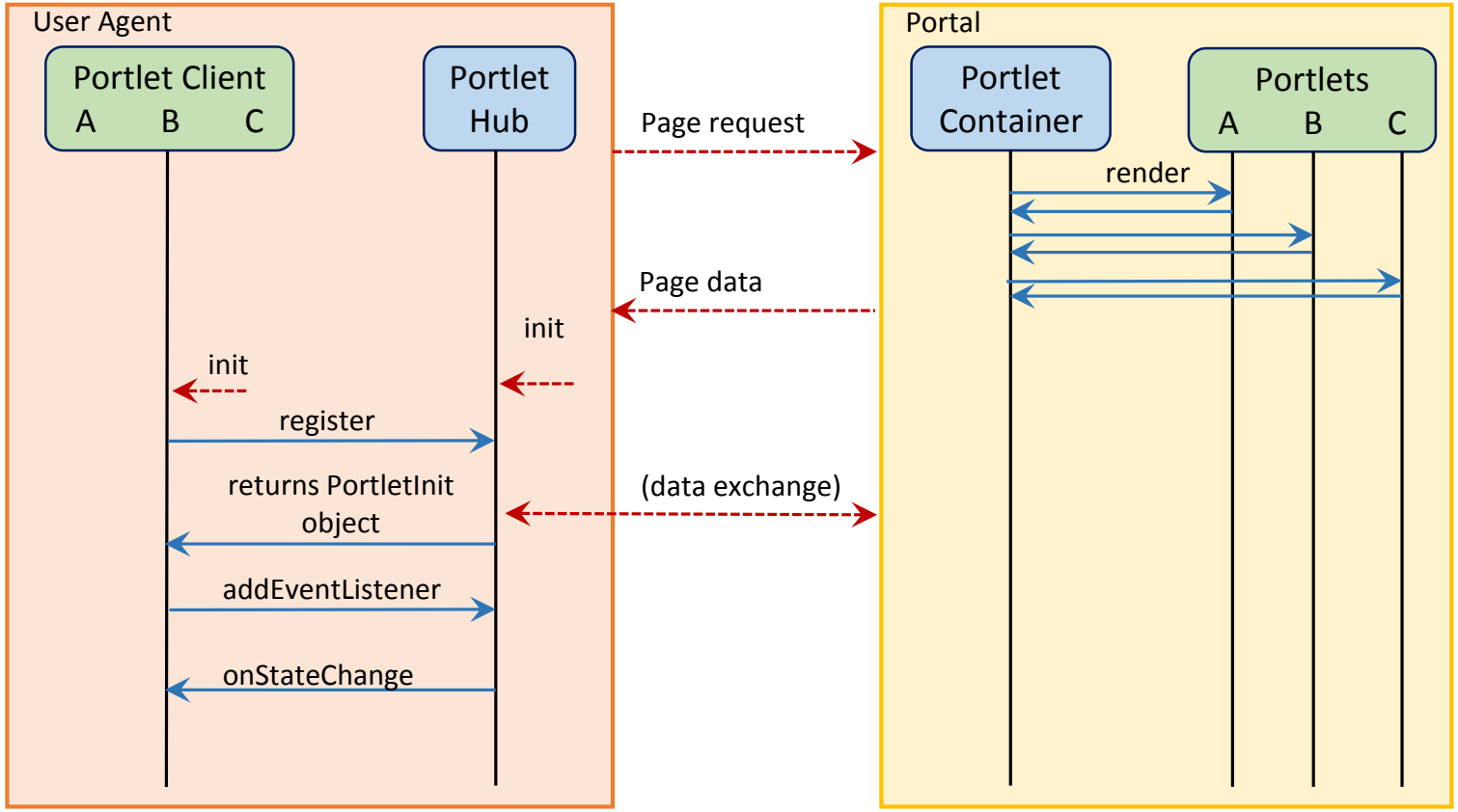

```
@ServeResourceMethod(portletNames="BeanPortletDemo",
                     resourceID="getImage")
public String imageMarkupMethod() {
   StringBuilder txt = new StringBuilder(128);
   String name = nameBean.getName();
   ...
   txt.append("<img src='").append(path).append("'>");
   return txt.toString();
}
```

- Produces data during resource phase, triggered through resource URL
- A string resource identifier can be specified in annotation
- Resource (servlet, JSP) can be included after method execution
- Multiple resource methods for single resource phase execution allowed
- Simplified method signatures

The Portlet Hub API



Portlet Hub API – Registration Sequence



Portlet Hub API – Portlet Client Registration Code

v3

```
var pid = '<portlet:namespace/>',
      resdiv = '<portlet:namespace/>-image',
      oldname=null, hub,

  // Handler for onStateChange event
  update = function (type, state) {
     var newname = null;
     newname = state.getValue('name');
     if (newname && (newname !== oldname)) {
     ...
     }

     oldname = newname;
  };

  // Register portlet with portlet hub
  portlet.register(pid).then(function (pi) {
     hub = pi;

     hub.addEventListener("portlet.onStateChange", update);
  });
```

Portlet identifier is the unique namespace for the portlet window. In this case, obtained using a portlet JSP tag.

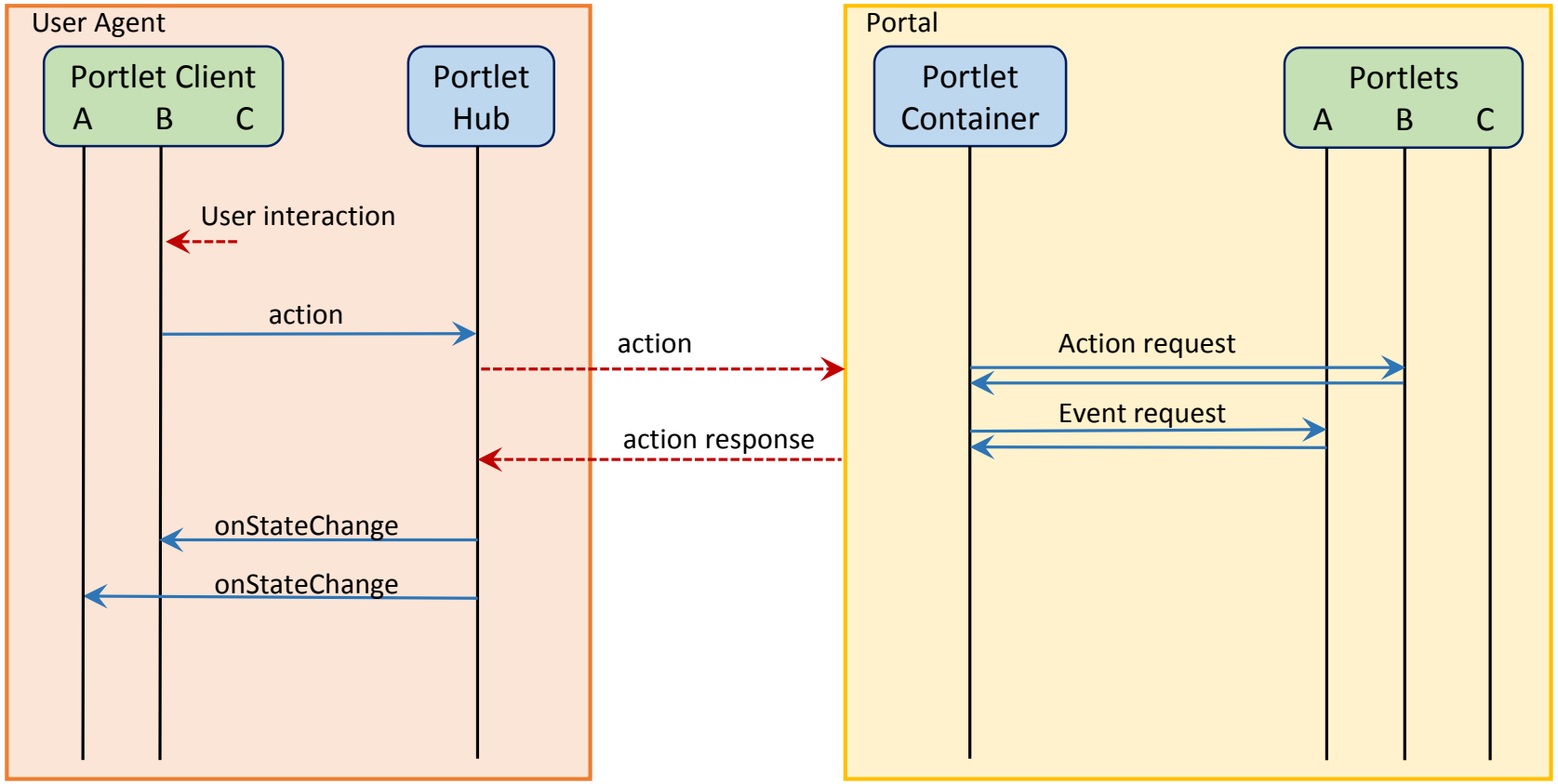
Portlet hub calls onStateChange event handler after registration and then when portlet state changes.

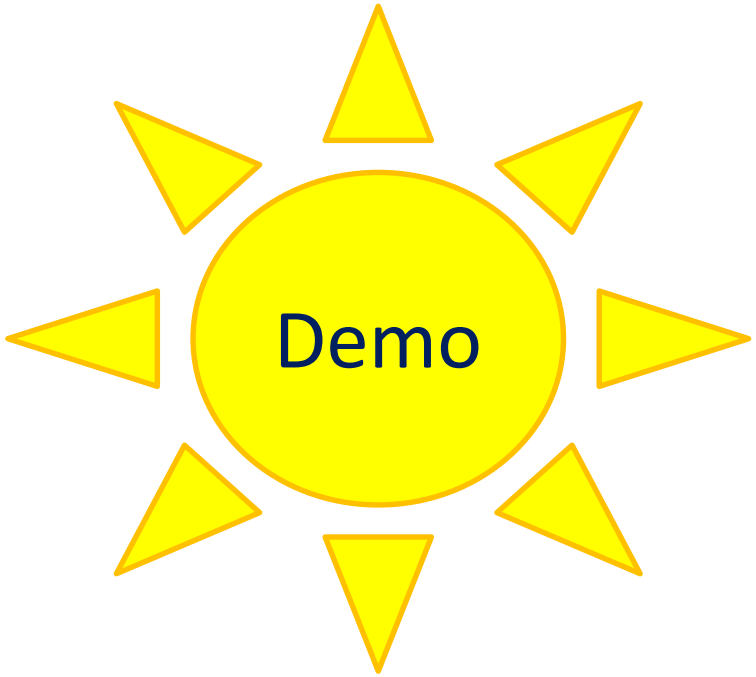
Portlet registers with portlet hub through function in global namespace.

The portlet hub provides a PortletInit object containing the portlet hub methods for use by the portlet.

The portlet registers the onStateChangeHandler

Portlet Hub Action Sequence





* Add the HelloWorldImage.java class
  * Uses injection to access a number of portlet artifacts
  * Defines an @HeaderMethod with portletNames="*"
    * Applies this header method to all portlets in the portlet application
    * Defines a page resource dependency for the portlet hub
      * Using the HeaderResponse#addDependency() method
    * The portlet container will place the portlet hub JavaScript API on the page
    * Dependency specification is a page resource ID
      * consisting of resource name, resource scope, and version
  * Defines an @RenderMethod using the include attribute
    * Ordinal 100, so rendered after the name but before the entry field
    * Causes the specified JSP to be included
    * The JSP contains JavaScript code that is used to load the image
  * Defines a @ServeResourceMethod with resourceID attribute set to "getImage"
    * The JavaScript portlet client addresses this resource method with an Ajax call
* Look at the helloWorldImage.jsp
  * Defines a div element to contain the image
    * The id attribute is based on the portlet namespace to make it unique on the page
  * Defines JavaScript code known as the portlet client
  * Defines update function as handler for the onStateChange event
    * Called with 2 arguments: event type and new portlet state
    * Portlet state consists of render parameters, portlet mode, and window state
  * If the name parameter has changed, the update function uses createResourceUrl method
    * Takes 3 arguments: resource parameters, cacheability option, and resource ID
    * Here, resource ID is set to 'getImage'; matches resource ID on serve resource method
    * Returns Promise object, which the portlet hub fulfils with the URL
    * The URL can be used like any other URL, for example, in an Ajax request
  * The portlet client registers with the portlet hub using the portlet.register() method
    * Takes the portlet namespace as argument, uniquely identifying the portlet
    * Returns a Promise object, which the portlet hub fulfills when it is ready
    * When it fulfills the promise, the portlet hub passes a PortletInit object
      * Contains functions and constants for portlet use
      * PortletInit object is specific to portlet at hand
  * The portlet hub registers an onStateChange listener with the portlet hub
    * Using the portlet hub addEventListener function
    * Takes 2 arguments: The event type, and the callback function, in this case, 'update'
* Add CSSIncludingHeaderMethod.java
  * Defines a single header method; applies to all portlets in the portlet application
  * Uses HeaderResponse#addDependency method to add a dependency
    * But this time, also provides source markup for that dependency
  * The portlet container will assure that the resource is only placed on the page once
  * The portlet container will select version if two resources differ only by version

JSR 378: Portlet 3.0 Bridge for JSF 2.2

- Building on JSR 329: Portlet Bridge for JSF 1.2
- New Reference Implementation (RI): Liferay Faces Bridge
- TCK Converted from JSP to Facelets
- TCK now utilizes <f:ajax/> instead of Trinidad Partial Page Rendering
- EG has voted +1 for almost all Bridge API issues
- Bridge API has been implemented and backported

JSR 378: Demo

- Single Page Application
- jsf.js client side library integration with Portlet 3.0 Hub

Portlet Configuration

- @PortletApplication for portlet application-level configuration

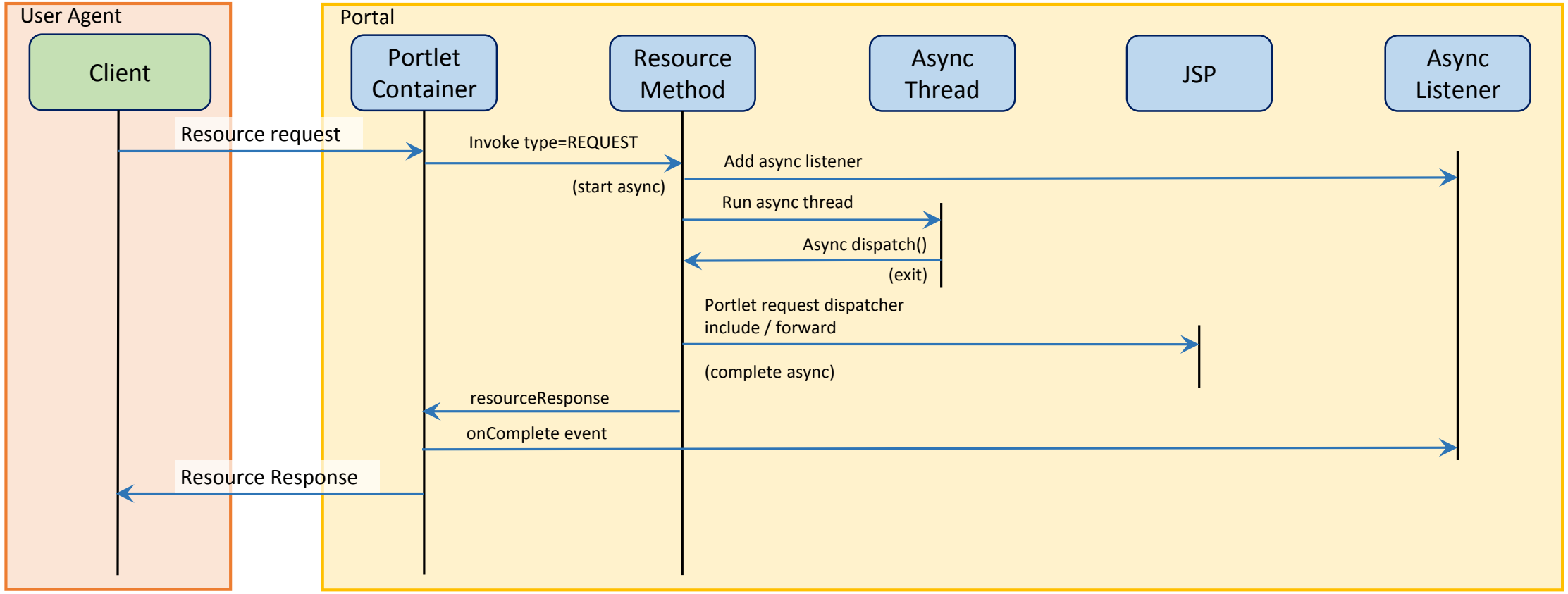
```
@PortletApplication(
      defaultNamespaceURI="https://www.java.net/",
      events = {
          @EventDefinition(qname=@PortletQName(namespaceURI="http://www.apache.org/",
                           localPart="event1"), payloadType = String.class),
          @EventDefinition(qname=@PortletQName(namespaceURI="",
                           localPart="event4"))
      }
)
```

- @PortletConfiguration for portlet-level configuration

```
@PortletConfiguration(portletName="Portlet1",
   initParams = {
      @InitParameter(name="color", value="#cafeba"),
   },
   title={
      @LocaleString(locale = "EN", value="Annotated Portlet"),
   }
)
```

Asynchronous Processing

- Example sequence, one of many possible



Asynchronous Processing – PortletAsyncContext Object


```
@Inject private ChatHistory history;
@Inject private ChatRoomRunner runner;
@Inject private ChatRoomListener listener;

@ServeResourceMethod(portletNames = "BeanPortletDemo",
                     asyncSupported = true,
                     resourceID="getChatHistory")
public void getChatHistory(ResourceRequest req, ResourceResponse resp)
      throws IOException, PortletException {
   boolean refresh =
      new Boolean(req.getResourceParameters().getValue("refresh"));

   PortletAsyncContext portletAsyncContext = req.startPortletAsync();
   portletAsyncContext.setTimeout(60000);
   portletAsyncContext.addListener(listener);

   runner.init(portletAsyncContext, refresh);
   portletAsyncContext.start(runner);
}
```

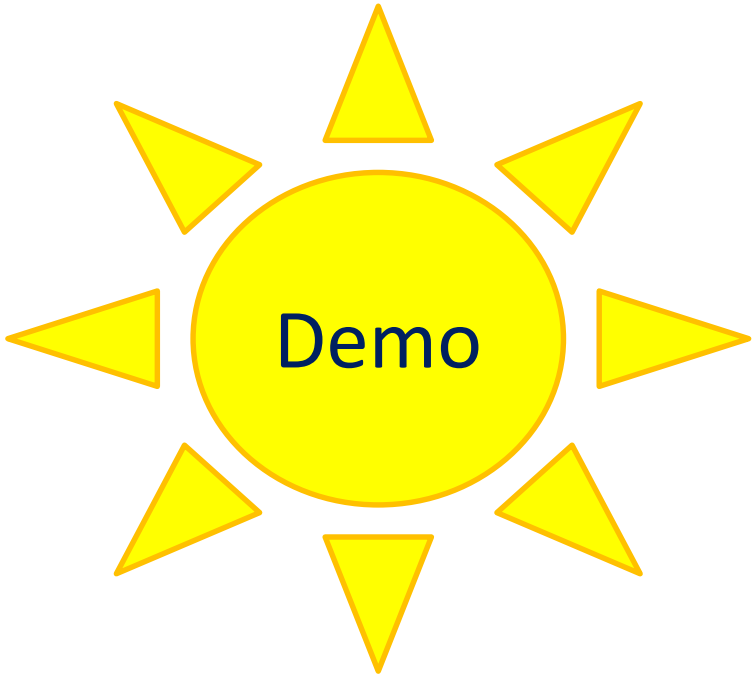
Dependencies are injected

Async supported is activated

Async processing is started

Async listener is registered

Async thread is started



* Look at ChatHistory.java @ApplicationScoped bean
  * Simply a container for message strings
  * Can add a message, clear messages, get markup for messages, get number of messages
    * Latter function is used to poll for new messages
  * Injects the NameBean to access the current name
* Look at ChatRoomListener.java
  * Implements an injectable PortletAsyncListener to check for timeout
  * If timeout occurs, a flag is set and the request is completed
* Look at ChatRoomRunner.java
  * An @PortletRequestScoped bean containing the 'run' method for async thread
  * Uses CDI injection
  * Thread is initialized and started during resource request processing
  * Initialized with the PortletAsyncContext object - needed for async processing
* The run() method polls the chat history until a change in # messages occurs
    * When change occurs, returns the markup through the ResourceRequest object
    * If the async listener shows that a timeout occurred, the run() method exits
    * Completes asynchronous processing using the PortletAsyncContext#complete() method
* Look at ChatRoom.java
  * Class that contains portlet lifecycle methods for the chat room
  * Defines @RenderMethod to render the chatroom
    * Serves a JSP containing the portlet client for Ajax updates
  * Defines @ServeResourceMethod that starts asynchronous processing to serve updates
    * Activates asynchronous processing using the annotation asyncSupported attribute
    * Shows how to obtain a resource parameter in the V3 manner
    * Starts async processing by calling the ResourceResponse#startPortletAsync() method
    * startPortletAsync returns a portletAsyncContext object
    * Sets the request timeout through the PortletAsyncContext#setTimeout() method
    * Adds an async listener to check for a timeout
    * Starts the async thread through the PortletAsyncContext#start() method
  * Defines 2 action methods to handle user input
    * actionName="addMessage" - used when message entered
    * actionName="clearHistory" - used when 'clear' button clicked
* Look at chatroom.jsp
  * Renders box containing the chat history
  * Renders form for user input
  * Provides portlet client JavaScript code
    * Defines getChat function, which constantly loops to get new chat history
      * The resource URL is created with resource ID 'getChatHistory' to address appropriate resource method
* Attaches onsubmit handler to form
      * The form is submitted through the portlet hub action method
    * Attaches onclick handler to the clear button
      * Instead of submitting the form, specific action parameters are submitted
      * In this case, the action name 'clearHistory' causes the server-side clearHistory action method to be called

More Items ....

- Multipart form support
  - For file uploads, etc.
- Fragment identifier (named anchor) for render URLs
- Ability to directly set HTTP status code for resource requests
- Static configuration for page resources
  - annotation or deployment descriptor
- Easily obtain the user agent string during any request
- Render parameter copy option when creating URLs

Thank You!

Further Information:

- JSR 362 Portlet Specification 3.0 JCP Page
  - https://www.jcp.org/en/jsr/detail?id=362
- JSR 362 Portlet Specification 3.0 Project Page
  - https://java.net/projects/portletspec3
- Apache Pluto Project (RI & TCK)
  - http://portals.apache.org/pluto/
  - https://github.com/apache/portals-pluto

Martin Scott Nicklous | Scott.Nicklous@de.ibm.com
Neil Griffen | neil.griffen@liferay.com