3

Bridge Interface

The `Bridge` interface is the main abstraction of the Portlet Bridge API. It defines the activation and invocation APIs whereby a portlet delegates processing to Faces. The bridge's lifecycle is similar to the portlet's lifecycle. Before executing requests, the bridge must be initialized and once the bridge becomes inactive it is destroyed. In a portlet application each portlet using the bridge's services maintains a distinct bridge instance. Typically, the portlet initializes the bridge in the portlet `init()` method and destroys it in the portlet `destroy()` method. However the only requirements are:

- the portlet must initialize the bridge before having it process a portlet Faces request.
- that once initialized, the portlet must not reinitialize the bridge until after its current use has been destroyed.
- that once destroyed, a bridge must not be reused to process a portlet Faces request until it has been initialized again.

3.1 Discovering and Instantiating the Bridge

The bridge is a pluggable component. The specific bridge implementation used by a portlet in a given deployment is controlled by the portlet. `Bridge` implementations are required to have null constructors. A portlet instantiates the designated bridge using the new operator. It is recommended that the portlet allow deployers the flexibility for designating the specific bridge this portlet uses in the given deployment environment. See 4.1 for a description on the techniques used by the `GenericFacesPortlet`.

A bridge must publish the specification name and version that it implements such that clients wishing to determine the specification version implemented by the bridge can get it from the package version information of any class in the API package (`javax.portlet.faces`):

```
Class c = Class.forName("javax.portlet.faces.Bridge");
String name = c.getPackage().getSpecificationTitle();
String version = c.getPackage().getSpecificationVersion();
```

For this specification, the returned name must be "Portlet 2.0 Bridge for JavaServer Faces 1.2"[3.1]. Its version number must correspond to the version

of this specification that this bridge implements. The version number contains 2 places separated by a "." (e.g. 1.0). The first number signifies the major version, the second number signifies the minor (maintenance) version.

3.2 Initializing the Bridge

A bridge is initialized by calling its init method:

```
public void init(javax.portlet.PortletConfig config) throws BridgeException;
```

Once initialized, a bridge is bound to the portlet that initialized it until it has been released. I.e. an initialized bridge can't be shared between differing portlets in the same application[nt].

The bridge initializes itself on the basis of application initialization parameters configured in the web.xml and `PortletContext` attributes set by the calling portlet. The bridge recognizes the following application initialization parameters:

```
javax.portlet.faces.MAX_MANAGED_REQUEST_SCOPES
javax.portlet.faces.RENDER_POLICY
javax.faces.LIFECYCLE_ID
```

`MAX_MANAGED_REQUEST_SCOPES` is an `Integer` valued configuration parameter that describes the maximum number of bridge request scopes maintained by the bridge at any given time for all the portlets in this web application[nt]. If not set the bridge provides an implementation dependent default maximum.

`RENDER_POLICY` is a `javax.portlet.faces.Bridge.BridgeRenderPolicy` (enum) valued configuration parameter that when set controls whether or not the bridge delegates view rendering to another handler. Valid values include: `ALWAYS_DELEGATE`, `NEVER_DELEGATE`, `DEFAULT`. The value `ALWAYS_DELEGATE` indicates the bridge should not render the view itself but rather always delegate the rendering[3.2]. The value `NEVER_DELEGATE` indicates the bridge should always render the view itself and never delegate[3.3]. The value `DEFAULT` indicates the bridge should follow the requirements as specified 6.2 concerning implementing `renderView()`. This section directs the bridge to first delegate the render and if and only if an `Exception` is thrown then render the view based on its own logic[3.4]. If the configuration parameter is not present or has an invalid value the bridge renders using default behavior[3.5]. I.e. as if `DEFAULT` is set.

`LIFECYCLE_ID` is a String valued configuration parameter that describes the `ID` of the `Lifecycle` the bridge uses when executing Faces requests[3.6]. If not set the bridge uses the ID for the default lifecycle (`LifecycleFactory.DEFAULT_LIFECYCLE`)[3.7].

In addition the portlet can impact a bridge's initialization by setting specific `PortletContext` attributes prior to calling `init()`. Because `PortletContext` attributes are application wide, to carry initialization parameters on a per portlet basis, the attribute name is encoded with the portlet name as described below.

The defined attributes are:

```
javax.portlet.faces.[portlet name].excludedRequestAttributes
javax.portlet.faces.[portlet name].preserveActionParams
javax.portlet.faces.[portlet name].defaultViewIdMap
javax.portlet.faces.[portlet name].bridgeEventHandler
javax.portlet.faces.[portlet name].bridgePublicRenderParameterHandler
javax.portlet.faces.[portlet name].defaultRenderKitId
```

where `[portlet name]` is the name of the portlet returned from a `PortletConfig.getPortletName()`.

`excludedRequestAttributes` is an attribute whose value is a `List` of `String` objects each of which either defines a specific attribute name the bridge is to exclude from its managed request scope or defines a set of attributes the bridge is to exclude from its managed request scope[3.8]. The later is identified by a `String` value with the wildcarded suffix ".*". Such wildcard usage indicates the bridge is to exclude all those attributes within the namespace identified by removing the "*"[3.9].

`preserveActionParams` is a `Boolean` valued attribute that when `TRUE` indicates the bridge must maintain the action's request parameters for the duration of the bridge request scope[3.10] 5.1.2. When this attribute isn't present or is `FALSE` the action's request parameters are only maintained for the duration of the portlet request scope. The exception to this is the `ResponseStateManager.VIEW_STATE_PARAM` parameter which is always maintained in the bridge request scope regardless of this setting[3.11].

`defaultViewIdMap` is a `Map<String, String>` valued attribute containing one entry per supported `PortletMode`. The `Map` key is the `String` name of the `PortletMode`. The `Map` value is the default Faces `viewId` for the mode[3.12].

`bridgeEventHandler` is a (instanceof) `BridgeEventHandler` valued attribute identifying the instance that is called to handle portlet events received by the bridge[3.20].

`bridgePublicRenderParameterHandler` is a (instanceof) `BridgePublicRenderParameterHandler` valued attribute identifying the instance that is called to postprocess incoming public render parameter changes after the bridge has pushed these values into the mapped managed beans. This handler gives the portlet an opportunity to recompute and get into a new consistent state after such changes[3.20].

`defaultRenderKitId` is a String valued attribute identifying the renderkit id the bridge should encode as a request parameter in each request to ensure that Faces will use this Id when resolving the renderkit used by this portlet[3.22]. If this attribute doesn't exist, the bridge takes no action, leaving the renderkit resolution to normal Faces resolution.

Non-standard attributes should follow a similar naming convention:

application wide attributes: attributes should be named `javax.portlet.faces.extension.`*[extension_name]*. Where *extension_pkg* is further namespacing introduced by the extension authors to avoid collisions with other (types of) extensions.

portlet specific attributes: attributes should be named `javax.portlet.faces.extension.`*[extension_name].[attribute name]* where *portlet name* is the name of the portlet returned from a `PortletConfig.getPortletName()` call and *attribute name* is the name of the attribute. Using *portlet name* ensures the bridge can locate the correct value on a per portlet basis.

3.3 Destroying the Bridge

The bridge is destroyed by calling its destroy method:

```
public void destroy();
```

When a portlet no longer needs the bridge's services, the portlet releases the bridge by calling the `destroy()` method. Typically this is done in the portlet's own `destroy()` method. Once `destroy()` has been called, no further requests can be processed through this bridge until a subsequent `init()` has occurred[3.13].

This call performs no action if the bridge is in an uninitialized state[3.14].

3.4 Request Processing

The bridge is used to execute Faces requests on behalf of the portlet by calling its `doFacesRequest` method:

```
public void doFacesRequest(
    javax.portlet.ActionRequest request,
    javax.portlet.ActionResponse response)
     throws BridgeUnititializedException, BridgeDefaultViewNotSpecifiedException,
            BridgeException, NullPointerException;

public void doFacesRequest(
    javax.portlet.RenderRequest request,
```

```
    javax.portlet.RenderResponse response);
     throws BridgeUnititializedException, BridgeDefaultViewNotSpecifiedException,
            BridgeException, NullPointerException;

public void doFacesRequest(ResourceRequest request, ResourceResponse response)
        throws BridgeUninitializedException,
            BridgeException;

public void doFacesRequest(EventRequest request, EventResponse response)
        throws BridgeUninitializedException,
            BridgeException;
```

Portlets call the appropriate form of this method for each of the request phases in the portlet lifecycle. In Portlet 2.0 the lifecycle can be more complicated than in Portlet 1.0. I.e. a typical Java Portlet Specification 1.0 (JSR 168) portlet lifecycle invokes the portlet `processAction()` and `render()` methods which results in two calls to the bridge's `doFacesRequest()`. The first processes the action request by passing the corresponding `ActionRequest` and `ActionResponse` objects and the second the render request by passing the corresponding `RenderRequest` and `RenderResponse` objects. Likewise, when a Java Portlet Specification 1.0 (JSR 168) portlet lifecycle only invokes `render()` such as on the initial request or as a result of `renderURL` invocation, `doFacesRequest()` is only executed once passing the corresponding `RenderRequest` and `RenderResponse` objects.

Though this lifecycle remains for Portlet 2.0, additional phases can occur/replace the Portlet 1.0 phases. Portlet events are communications sent to a portlet by the consuming application to convey pertinent information when interactions occur outside of this portlet. i.e. if a portlet action is the lifecycle phase executed when a user directly interacts with the portlet, a portlet event is the lifecycle phase when an interaction outside of this portlet causes the consumer to need to inform this portlet of the change. Because the portlet model calls for all events (stemming from a client interaction) to be completed before rendering occurs and event processing can trigger new events to be raised, a portlet can either receive the event as a side effect of it having raised an event during its own action processing or in the more common form of the action being directed elsewhere.

The other new request lifecycle phase in Portlet 2.0 is the resource serving phase. Whereas in Portlet 1.0 resources couldn't be directly served by the portlet but rather had to be referenced via direct http references, Portlet 2.0 allows a portlet to directly return dependent resources. One common use case for this new lifecycle is to implement "rich client" behaviors using technologies such as `AJAX`. In this use case the resource is a portion of the portlet's page markup which the client requests and inserts directly. Whether this or the more convention use case of acquiring a dependent resource such as a javascript file

or image, the resource phase can be thought of as an additional step that is part of but after a render phase.

Each of the `doFacesRequest` lifecycle phase methods takes a portlet request and response object. In addition to the incoming information in the portlet request, the portlet provides additional bridge request context by setting attributes in the request using `PortletRequest.setAttribute()`. To direct the bridge to execute a specific Faces target the portlet can set either of the following before calling `doFacesRequest`:

`javax.portlet.faces.viewId`: The value of this attribute identifies the Faces `viewId` the bridge must use for this request (e.g. `/myFacesPage.jsp`). This is expected to be a valid Faces viewId though it may optionally contain a query string[3.15].

`javax.portlet.faces.viewPath`: The value of this attribute contains the the Faces `viewId` the bridge must use for this request in `ContextPath` relative path form (e.g. `/faces/myFacesPage.jsp`). This value may optionally contain a query string[3.16].

The `BridgeUninitializedException` is thrown if this method is called while the bridge is in an uninitialized state[3.17]. The `NullPointerException` is thrown if either the passed request or response objects are null[3.18].

Previous

Portlet 2.0 Bridge for JavaServerTMFaces 1.2 – November 14th, 2010

Next