7

Supporting Faces WriteBehindResponse

In JavaServer™ Faces 1.2, a JSP is the primary page format for representing a Faces view. Faces provides a set of taglibs which developers reference to describe their view. When a JSP is rendered, the Faces tag implementations are invoked which constructs the view tree. After JSP processing is complete, control returns to Faces which then renders the Faces markup by rendering the view tree. Because JSP rendering and Faces (view) rendering occur at different stages of the rendering phase, any output rendered within the JSP itself (literal strings/static html or jsp/java rendered markup) will be output before the Faces view markup regardless of whether it logically appears before of after the Faces view description in the JSP. To preserve the natural order in a JSP, Faces 1.2 utilizes an implementation dependent write behind mechanism which generally uses the following pattern:

- Defines a response wrapper class/interface that includes implementation dependent APIs.
- Implements its Faces `view` tag handler (and maybe others) to check if the current response object (on the `ExternalContext`) supports the implementation dependent API, and if so call the method to flush any existing (prior JSP) output to the real response stream before processing the tag and building the Faces `view`.
- Implements its `ViewHandler`, to replace the existing `ExternalContext` response object with with one that implements the extended implementation dependent API before dispatching to the JSP. And then once the dispatch has returned, it restores the prior response object to the `ExternalContext`, renders the Faces `view` and then calls another of the implementation dependent APIs on the response object it dispatched with to output any remaining (buffered) markup.

Because the JSF specification prescribes this behavior without formalizing the mechanism, the bridge must. The bridge defines two formal mechanisms. One, relies on a `ServletFilter` and was defined for the Portlet 1.0 Bridge because `PortletResponseWrappers` weren't part of the portlet 1.0 specification. Its support continues for backwards compatibility and for those Faces implementations where the implementation of the bridge's second mechanism isn't feasible. The second, mirrors the pattern described above using a `PortletResponseWrapper` that implements a required bridge write behind interface which supports and works in conjunction with the Faces implementation dependent API needed by the (`view`) tag handlers.

The bridge is required to support both of these mechanisms simultaneously in its provided ViewHandler [6.2.1].

7.1 Support via a ServletFilter

The bridge's `ServletFilter` mechanism is for use when either the Faces implementation's mechanism can't be incorporated into the bridge's wrapper mechanism or to continue (backwards compatibility) support from the Portlet 1.0 Bridge when portlet wrappers didn't exist. The bridge's responsibility is as follows:

- Prior to dispatching to the JSP, the bridge's `ViewHandler` sets the `javax.portlet.faces.RenderContentAfterView` attribute with a boolean value of `Boolean.TRUE`.
- After the dispatch completes and the `ViewHandler` has rendered the Faces view, the `ViewHandler` gets and outputs any write behind markup be reading the request attribute `javax.portlet.faces.AfterViewContent` (containing either a `byte[]` or `char[]`)

To enable and use this mechanism, the portlet developer implements and configures a `ServletFilter` that has the following behavior:

- When invoked, if the `javax.portlet.faces.RenderContentAfterView` attribute exists and has a value of `Boolean.TRUE`, wrap the incoming `ServletResponse` with either the implementation dependent `ServletResponseWrapper` used by the particular Faces implementation being used or one that supports its required API and behavior. Replace the `ExternalContext response` object with this wrapper caching the current one and then call the filter chain to execute the render.
- Upon completion, restore the `ExternalContext response` object, get the after view content from the wrapped response using its proprietary API, and write this content (as a `byte[]` or `char[]`) as the value of the request attribute `javax.portlet.faces.AfterViewContent`.

7.1.1 Example Configuration

As a `ServletFilter`, the write behind filter is configured in the application's `web.xml`. It should only be configured to filter included requests:

```
<filter>
    <filter-name>myFacesRenderBehindPortletFilter</filter-name>
   <filter-class>org.apache.myfaces.portlet.faces.application.BridgeMyFacesRenderFilter</
</filter>
```

```
<filter-mapping>
    <filter-name>myFacesRenderBehindPortletFilter</filter-name>
    <url-pattern>*.jspx</url-pattern>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>

<filter-mapping>
    <filter-name>myFacesRenderBehindPortletFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

7.1.2 Version Considerations

The following Faces implementation versions implement its dependent write behind behavior exclusively in a `ServletResponseWrapper` without independently defining a non-Servlet dependent interface:

- Mojarra 1.2_03
- All released version of MyFaces 1.2 (current version is 1.2.9)

I.e. the Faces view tag handler(s) test to see if the response supports its write behind behavior involves checking whether the response object is an instance of the particular `ServletResponseWrapper` it has implemented.

Because, the bridge's write behind wrapper mechanism requires the behavior be implemented in an object that both extends `PortletResponseWrapper` and implements the `BridgeWriteBehindResponse` interface, this mechanism is incompatible with the above versions. If the target runtime environment will contain any of these versions, one must use this servlet filter mechanism to support the write behind behavior.

7.1.3 Performance Considerations

Through the use of the request attribute to signal the filter to introduce its support, the mechanism is designed to have negligible impact when invoked from non-portlet requests. However, some overhead exists because the Servlet container has to determine whether this filter should be called (i.e. if its an include) and if invoked whether the filter needs to act (based on existence/setting of the `javax.portlet.faces.RenderContentAfterView` attribute. In addition, because this facility isn't provided by default; setting up the filter requires extra configuration. Because many JSF/JSP pages aren't coded in a manner that exhibits this ordering problem, portlet developers should keep the above in mind when deciding whether enabling the filter in their application is justified.

7.2 Support via Portlet 2.0 ResponseWrapper

The bridge's `PortletResponseWrapper` mechanism follows a pattern similar to Faces `ServletResponseWrapper` mechanism. The bridge has the following responsibility:

- In the bridge's `ViewHandler`, prior to dispatching to the JSP, replace the existing `ExternalContext response` with one that subclasses the appropriate phase's `PortletResponseWrapper` class and implements the bridge's `javax.portlet.faces.BridgeWriteBehindResponse` interface.
- Dispatch to the JSP.
- Restore the prior `ExternalContext` response
- Call the wrapper's `hasWriteBehindMarkup()` method and if it returns `true` and something hasn't already set the `javax.portlet.faces.AfterViewContent` request attribute then hold onto the wrappers content for later output. Otherwise, write its buffered content to the response.
- Render the Faces view.
- Output the (cached) after view content.

For this mechanism to work, the `PortletResponseWrapper` must additionally support the particular Faces dependent API used by the (view) tag handler to flush the prior view content. The particular wrapper used by the bridge is configured by the portlet in `faces-config.xml`. In the absence of such configuration, the bridge may use its own response wrapper that is designed to work automatically in the Faces environments it targets. For example, where Faces write behind implementations exist that have their tag handlers rely on Java reflection to determine support, the implementation of the default response wrapper should support any such additional APIs as needed by these implementations to allow these classes to be used directly in the above mechanism and achieve write behind behavior.

7.2.1 Configuring the Bridge to use a PortletResponseWrapper/WriteBehindResponse implementation

One configures the particular implementation of the `RenderResponseWrapper` and/or `ResourceResponseWrapper` the bridge uses as the response object it dispatches to in the bridge's `application-extension` section of the `face-config.xml`[7.1]. The syntax for this is described in `portlet2.0-bridge-faces1.2-faces-config-extensions.xsd`.

```
<application>
    <application-extension>
        <bridge:write-behind-response-wrappers>
            <bridge:render-response-wrapper-class>
```

```
                org.mypackage.MyWriteBehindRenderResponseWrapper
            </bridge:render-response-wrapper-class>

            <bridge:resource-response-wrapper-class>
                org.mypackage.MyWriteBehindResourceResponseWrapper
            </bridge:resource-response-wrapper-class>
        </bridge:write-behind-response-wrappers>
    </application-extension>
</application>
```

7.2.2 Version Considerations

Mojarra version 1.2_03 tag handlers detect write behind support by using `instanceof` a particular `ServletResponseWrapper` class. This class is incompatible with this `PortletResponseWrapper` mechanism, hence only the `ServletFilter` mechanism can be used to add write behind support.

Mojarra version 1.2_04 through 1.2_07 tag handlers detect write behind support by using `instanceof` a particular interface (`InterleavingResponse`) the response object might additionally implement. In such an environment, developers need to implement and configure appropriate `PortletResponseWrapper` classes that additionally implement both the `BridgeWriteBehindResponse` interface and the `InterleavingResponse` interface.

Mojarra version 1.2.08 through current (1.2_13) tag handlers detect write behind behavior by using Java reflection to detect if the response class implements the needed method. The supported methods correspond to those methods defined in the `InterleavingResponse` interface. The bridge should implement and use default `PortletResponseWrapper` classes that not only implements the `BridgeWriteBehindResponse` interface but also supports the methods defined in the `InterleavingResponse` interface. In such circumstances, because reflection is being used, the bridge can continue to be Faces implementation independent yet still automatically support write behind behavior when run in one of these versions. If a particular bridge doesn't naturally support these methods, a developer can enable write behind support by configuring the application with the same write behind `PorttletResponseWrapper`(s) that work for versions 1.2_04-1.2_07.

All (existing) MyFaces versions (1.2.2-1.2.9) tag handlers detect write behind support by using `instanceof` a particular `ServletResponseWrapper` class. This class is incompatible with this `PortletResponseWrapper` mechanism, hence only the `ServletFilter` mechanism can be used to add write behind support.

Previous

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

Next