Portlet-to-portlet communication between JSR 168 portlets on public pages

Richard Scott
Hongqing Song

April 05, 2006

Introduction

The WebSphere® Portal V5.1.0.1 built-in support for portlet-to-portlet communication between JSR 168 portlets, called cooperative portlets, uses the property broker. This support is for authenticated users; it does not currently work for anonymous users accessing public pages. Readers of our earlier articles (see Resources), as well as requirements for several of our current projects, have lead us to address this latter scenario.

Two solutions we have developed to support anonymous users accessing public pages involve:

1. Using application scope session
2. Using the WebSphere Portal programming model Systems Programming Interfaces (SPI)

This article is for architects and developers who need to design or implement portlet-to-portlet communication between JSR 168 portlets. Readers should have basic experience in portlet development using WebSphere Portal and Rational® Application Developer. For an overview of developing portlets, see the IBM Redbook IBM Rational Application Developer V6 Portlet Application Development and Portal Tools.

About the sample code

You can download the sample code for these two implementations and refer to the code as you read the rest of this article.

About the scenarios

Similar to the scenarios we described in our earlier articles, we use the "Quick search" example to illustrate these two approaches. Suppose a company creates a "Quick search" portlet using the JSR 168 API, and that this portlet can be used to search resources on the company's Web site. The portlet is placed on public pages (that is, on pages that do not require users to log in). When a
user inputs search text or clicks a link in the "Quick search" portlet, the search text string is passed to the "Search results" portlet, which does the search and displays the results. So, this scenario involves portlet-to-portlet communication between JSR 168 portlets ("Quick search" and "Search results") which are on public pages.

Figure 1 shows the two portlets in the first scenario. The communication between the portlets is implemented using application scope session. A user enters the search string "WebSphere Portal", and then clicks the Submit button; the "Search result" portlet receives the string and displays the result (in this simplified case, just a statement indicating that a search engine could be called).

**Figure 1. Scenario 1: Portlet-to-portlet communication using application scope session**

![Image of the two portlets](image)

A user enters the search string "WebSphere Portal", and then clicks the Submit button; the "Search result" portlet receives the string and displays the result (in this simplified case, just a statement indicating that a search engine could be called).

Figure 2 shows the second scenario in which the communication between the two portlets is implemented using the Model SPI. The user interface is the same as the first scenario; just the communication mechanism is different.
Figure 2. Scenario 2: Portlet-to-portlet communication using the Model SPI

The following products were used to develop and test the two sample portlets:

- IBM WebSphere Portal V5.1.0.1
- IBM Rational Application Developer for WebSphere V6.0, with WebSphere Portal V5.1 test environment
- Microsoft® Internet Explorer V6.0
- Netscape v7.1
- Mozilla Firefox V1.0

Scenario 1. Portlet-to-portlet communication using application scope session

The portlet session object provides session tracking for all requests coming from the same client. All portlets in the same portlet application share a portlet session for the user. Portlets in other portlet applications cannot access this session object; therefore, they cannot share attributes.

PortletSession provides two static variables to be used when adding attributes. The two variables representing the scopes are APPLICATION_SCOPE and PORTLET_SCOPE. Objects stored in the APPLICATION_SCOPE are shared among all portlets in the same portlet application. Attributes stored as PORTLET_SCOPE are available only to the portlet.

In WebSphere Portal, you can enable public session for anonymous users, and then package portlets that need to communicate with each other in the same portlet application. Then they can talk to each other by using application scope session.
In Listing 1, the JSP file is the user interface for the "Quick search" portlet (see Figure 1). When a user inputs search text and clicks the Submit button, or clicks the link, a portlet action is generated against the "Quick search" portlet.

**Listing 1. QuickSearchPortletView.jsp**

```jsp
<%@ page session="false" contentType="text/html"
    import="java.util.*,javax.portlet.*,p2p6.quicksearch.portlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<DIV style="margin: 6px">
  <P>
    PortletURL actionUrl = renderResponse.createActionURL();
    actionUrl.setParameter(QuickSearchPortlet.ACTION_NAME_PARAM,
      QuickSearchPortlet.P2P_QUICK_SEARCH_ACTION);
  </P>

  Please enter search text:
  <FORM method="POST" action="<%= actionUrl.toString() %>">
    <INPUT name="<%=QuickSearchPortlet.FORM_TEXT%>" type="text"/>
    <INPUT name="<%=QuickSearchPortlet.FORM_SUBMIT%>" type="submit" value="Submit"/>
  </FORM>

  <P>
    actionUrl.setParameter(QuickSearchPortlet.FORM_TEXT, "WebSphere Portal");
  </P>
  <a href="<%=actionUrl.toString()%>">Search resources for WebSphere Portal</a>

</DIV>
```

**Listing 2. The "Quick search" portlet processAction() and getSessionBean() methods**

```java
public void processAction(ActionRequest request, ActionResponse response) throws PortletException, java.io.IOException
{
    if( P2P_QUICK_SEARCH_ACTION.equalsIgnoreCase(request.getParameter(ACTION_NAME_PARAM)) )
    {
        // Get the session bean
        QuickSearchPortletSessionBean sessionBean = getSessionBean(request);
        // Set the search text in the session bean
        if( sessionBean != null )
        {
            sessionBean.setFormText(request.getParameter(FORM_TEXT));
        }
    }
}

private static QuickSearchPortletSessionBean getSessionBean(PortletRequest request)
{
    //Get portlet session
    PortletSession session = request.getPortletSession();
    if( session == null )
    {
        return null;
    }
    //Get the session bean from the application scope session
    QuickSearchPortletSessionBean sessionBean = (QuickSearchPortletSessionBean)session.getAttribute(SESSION_BEAN,
     portletSession.APPLICATION_SCOPE);
    if( sessionBean == null )
    {
        //If no session bean in the session, create one and put it in application scope session so that it
        //can be accessed by other portlets in the same portlet application.
        sessionBean = new QuickSearchPortletSessionBean();
    }
```

Portlet-to-portlet communication between JSR 168 portlets on public pages
Listing 3 shows how the "Search result" portlet retrieves the search string which was set by the "Quick search" portlet. The portlet gets the session bean from the portlet session with application scope, and then retrieves the search string in the session.

Listing 3. The "Search result" portlet doView() and getSessionBean methods

```java
public void doView(RenderRequest request, RenderResponse response) throws PortletException, IOException {
  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  // Get the session bean
  QuickSearchPortletSessionBean sessionBean = SearchResultPortlet.getSessionBean(request);
  String searchText = "";
  if (sessionBean != null && sessionBean.getFormText() != null && !"".equals(sessionBean.getFormText())) {
    // Retrieve the search text from the session bean
    searchText = sessionBean.getFormText();
    // Clean up the session bean
    sessionBean.setFormText("");
  }

  request.setAttribute(SEARCH_TEXT, searchText);
  // Invoke the JSP to render
  PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(VIEW_JSP);
  rd.include(request, response);
}

public static QuickSearchPortletSessionBean getSessionBean(PortletRequest request) {
  PortletSession session = request.getPortletSession();
  if (session == null) {
    return null;
  }

  // Get the session bean from portlet session with application scope
  QuickSearchPortletSessionBean sessionBean = (QuickSearchPortletSessionBean) session.getAttribute(SESSION_BEAN,
      PortletSession.APPLICATION_SCOPE);
  return sessionBean;
}
```

The sample code for scenario 1 is included in P2PSample6.war file in the downloaded.

Scenario 2. Portlet-to-portlet communication using the Model SPI

WebSphere Portal V5.1.0.1 includes built-in classes called the WebSphere Portal System Programming Interface (SPI), which enable developers to create an URL that points to a portal page and passes data to JSR portlets on that page. See LINK to Resources for access to the SPI Javadoc and the article Exploiting the WebSphere Portal V5.1.0.1 programming model, Part 1: Introducing the model by Stefan Hepper and Stefan Liesche for a more detailed description of the Model SPI.

Figure 3 shows the solution using Model SPI. The process is as follows.

1. A user enters search text and clicks Submit. Alternatively, the user just clicks the static link on the "Quick search" portlet. Either way, the data is sent as HTTP parameters to handler.jsp.
2. The handler.jsp gets the HTTP parameters sent from the Web page. It passes these parameters to Java™ classes in P2PUtil.jar, which is created using the Model SPI.

3. The Java classes in P2PUtil.jar create the URL that points to the "Search result" portal page. Parameters sent from the "Quick search" portlet are included as render parameters in the URL.

4. The handler.jsp gets the URL and redirects to the "Search result" portal page. The "Search result" portlet retrieves the render parameters, and performs the search.

Figure 3: Component diagram for the Model SPI scenario

Listing 4 is the JSP file for the user interface for the "Quick search" portlet (see Figure 2); the source code is included in P2PSample7.war file. When a user enters a search text and clicks Submit button, or clicks a link in the portlet, the handler.jsp is invoked and the search string is sent to it.

Listing 4. QuickSearchPortletView.jsp

```jsp
<%@ page session="false" contentType="text/html"
import="java.util.*,javax.portlet.*,p2p8.quicksearch.portlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<DIV style="margin: 6px">
  <STRINGBUFFER urlBuffer = new StringBuffer("/wps/urlhandler/handler.jsp");
  %>
  Please enter search text:
  <FORM method="POST" action="<%= urlBuffer.toString() %>">
    <INPUT name="QuickSearchPortlet.FORM_TEXT" type="text"/>
    <INPUT name="submit" type="submit" value="Submit"/>
  </FORM>
  <STRINGBUFFER urlBuffer.append("?searchText=WebSphere Portal");
  %>
  <a href="<%=urlBuffer.toString() %>">Search resources for WebSphere Portal</a>
</DIV>
```

Listing 5 is the source code for the handler.jsp file. It receives the search string sent from "Quick search" portlet, creates a URL string pointing to the "Search result" portal page with the search string as a URL parameter, and then redirects to the "Search result" portal page.
Listing 5. The handler.jsp

```java
<%@ page import="p2phandler.util.P2PUrlGenerator" %>

// Retrieve HTTP parameter sent from legacy Web page.
String searchStr = request.getParameter("searchText");
try {
    // Call classes in P2PUtil.jar to create the URL.
    // Parameters sent from "Quick search" portlet are included in the URL.
    String targetURLStr = P2PUrlGenerator.generateUrlString(
        "searchText", searchStr);
    // Redirect to the "Search result" portlet page
    response.sendRedirect(targetURLStr);
} catch(Exception e) {
    System.err.println(e.toString());
}
```

Listing 6 shows the "Search result" portlet getting the search string from the URL parameter. The search string comes from the "Quick search" portlet.

Listing 6. The doView() method of the "Search result" portlet

```java
public void doView(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    // Set the MIME type for the render response
    response.setContentType(request.getResponseContentType());
    String searchText = "";
    if(request.getParameter(SEARCH_TEXT) != null && !"".equals(request.getParameter(SEARCH_TEXT))) {
        //Get the search string from the URL parameter.
        searchText = request.getParameter(SEARCH_TEXT);
        request.setAttribute(SEARCH_TEXT, searchText);
        // Invoke the JSP to render
        PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher( VIEW_JSP);
        rd.include(request, response);
    }
}
```

Listing 7 shows how to generate the URL with render parameters using the WebSphere Portal Model SPI classes. The unique name of the portal page and the unique name of the "Search result" portlet are required. These unique names are specified in the deployment file, importP2PSample.xml.

Listing 7. P2PURLGenerator.java class

```java
public class P2PURLGenerator {
    public static String generateURLString(
        String pageName, // Custom unique name of the page containing the portlet
        String portletName, // The unique name of the portlet window
        String paramName,  // The parameter name
        String paramValue  // The parameter value
    ) throws StateException, NamingException, IOException {
        final StateManager mgr = getStateManager();
    }
}
```
// Get the URL factory
final URLAccessorFactory URLFactory = (URLAccessorFactory) mgr.getAccessorFactory(URLAccessorFactory.class);

// Request a URL
// The 3rd argument specifies whether the URL points to the protected (authenticated) area;
// pass in "false" if your page is accessible for unauthenticated users
final EngineURL URL = URLFactory.newURL(new MyServerContext(), false, true, mgr.newState(), Constants.EMPTY_COPY);

// Set the page this URL should point to
final SelectionAccessorFactory selectionFactory = (SelectionAccessorFactory) mgr.getAccessorFactory(SelectionAccessorFactory.class);
// Request the selection controller to set the page; pass in the state associated with the created URL
final SelectionAccessorController selectionCtrl = selectionFactory.getSelectionController(URL.getState());
// Set the page; you need the unique name (String) or the ObjectID of that page
selectionCtrl.setSelection(pageName);
// Dispose the accessor (avoids memory leak)
selectionCtrl.dispose();

// Set portlet render parameters
final PortletAccessorFactory portletAccessorFactory = (PortletAccessorFactory) mgr.getAccessorFactory(PortletAccessorFactory.class);
// Get the portlet controller to set render parameters;
// pass in the state associated with rge created URL
final PortletAccessorController portletCtrl = portletAccessorFactory.getPortletController(portletName, URL.getState());
// Get the modifiable render parameter map
final Map parameters = portletCtrl.getParameters();
// Set the render parameter
parameters.put(paramName, new String[]{paramValue});
// Dispose the accessor (avoids memory leak)
portletCtrl.dispose();

// Now convert the URL to a String; pass in your writer.
// writeDispose() writes the URL to the given writer and disposes the URL afterwards.
// If you want to display this URL a multiple times pls use writeCopy().
return URL.writeDispose(new StringWriter()).toString();

Listing 8 show the file that is used to deploy the sample application. The unique names of the portal page and the "Search result" portlet are defined here.
Listing 8. The importP2PSample.xml file

...<content-node action="update" active="true" allportletsallowed="true" content-parentref="_6_081S465CDF0Q16NL_30U" create-type="explicit" objectid="_6_081S465CDF0Q16NL_341" ordinal="100" type="page" uniquename="p2p.test.page2">  
  <supported-markup markup="html" update="set"/>
  ...
  <component action="update" active="true" deletable="undefined" uniquename="p2p.test.page2.SearchResult" modifiable="undefined" objectid="_7_081S465CDF0Q16NL_5LD" ordinal="100" type="control" width="undefined">
    <portletinstance action="update" objectid="_5_081S465CDF0Q16NL_5BU" portletref="_3_081S465CDF0Q16NL_3GI" shareref="undefined"/>
  </component>
  ...
</content-node>
...

Deploying the sample code

To deploy the sample code:

1. Download the sample zip file, and unzip it.
2. Stop WebSphere Portal.
3. Place P2PSample6.war, P2PSample7.war, and P2PSample8.war in the
   ${wps}/installableApps directory
   where ${wps} means the WebSphere Portal installation directory.
4. Copy the xmlaccess directory to the ${wps}/bin directory.
5. Copy the urlhandler directory in ${was}/installedApps/server_name/wps.ear/wps.war
   where ${was} means the WebSphere Application Server installation directory. This is required
   by solution 2, and this directory has the handler.jsp file.
6. Open file ${wps}/shared/app/config/services/NavigatorService.properties, and set up
   "public.session = true". This is required by solution 1, which must enable public session
   for anonymous users.
7. Start WebSphere Portal.
8. Go to the ${wps}/bin/xmlaccess directory.
9. Run the command (on one line):

   ..\xmlaccess.bat -in importP2PSample.xml -out importResult.xml -user username
   -password password -url http://servername:9081/wps/config

10. Check the importResult.xml file to make sure the deployment is successful.
11. Create a custom url p2p_test_page1 that points to page:
   /p2p_test_label/P2P test page - APPLICATION_SCOPE_SESSION
12. Test the application using URL:
   http://localhost:9081/wps/portal/p2p_test_page1

Comparing the two solutions

Table 1. Each solution has its advantages and disadvantages.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>

Portlet-to-portlet communication between JSR 168 portlets on public pages
<table>
<thead>
<tr>
<th>1. Application scope session</th>
<th>Is easier to implement and deploy the application. No limitation for the size of the data passed between the portlets.</th>
<th>The portlets that need to communicate with each other must be packaged in the same WAR. There is a performance hit to the WebSphere Portal server to enable the public session for anonymous users.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Model SPI</td>
<td>The portlets that talk to each other do not have to be packaged in the same WAR. Enables portlets to communicate with each other on the same portal page as well as on different pages. In other words, this solution can also be used for page-to-page communication between JSR 168 portlets on public pages.</td>
<td>The length of a URL is limited. (2083 characters in Internet Explorer). If the size of the communication data is too big, the solution won’t work.</td>
</tr>
</tbody>
</table>

**Conclusion**

Portlet-to-portlet communication between JSR 168 portlets on public pages can be implemented using application scope session (Solution 1) or using Model SPI (Solution 2). Using the Model SPI enables portlets to communicate with each other on the same portal page as well as across portal pages (page-to-page communication).
Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
</table>
| Code samples  | 0604_scott-p2psamples.zip (HTTP | 26 KB

© Copyright IBM Corporation 2006

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)