Window scope

Table of contents

1. Introduction

Image that you have a bean in session scope. Suppose that, for some reason, the user opens more than one page. If you modify that bean in a window, its effect will be, obviously, reflected to all windows. Therefore, you could have some strange behaviour due to the fact that the other pages are not synchronized for the new value of the bean.

A typical case is a wizard. Suppose that the user is in the second page of the wizard, where the user had already put some information in the first page and it has been submitted into a bean in session scope. Then the user opens a link into a new window, then returns to the first page (with the wizard) and, depending on the action he did, maybe the bean changed its state. The wizard should not go on, because the data are not correct!

The token pattern can help, but it can help when the user presses the "back" button, not when some other action modifies the state of the web application, especially in another window.

This is the motivation on "finding" the window scope: each window has its own context, with its attributes.

2. What to put in window scope

Essentially, it is advisable to put in window scope all the beans that are useful to identify the current navigation state, for example for storing all the previous choices the user took.

This problem is typical of tree-based navigation. Normally if you select an element in the first level, then another in the second level, etc. to preserve the state you have two choices:

- put the choices in session scope: it can lead to inconsistencies when the user opens more windows;
- propagate the choices from request to request: it is difficult to achieve, because it involves more code to write.

Instead, using window scope, everything each element of the tree can be stored and retrieved for the next request (and page), preserving navigation even with more that one window.

3. What not to put in window scope

You should not put in window scope (and use session scope) things that must be shared during the entire session in all windows, for example:

- user information, such as username and credentials;
- the shopping cart, but the beans needed to add an item in the shopping cart can be put in window scope.

4. Using Scopes with "window" scope

To use the "window" scope support in your web application you have to:

- extract the binary release;
- copy the "scopes.jar" file in the directory `[webapp_root]/WEB-INF/lib`;
- copy "lib/commons-io.jar" in `[webapp_root]/WEB-INF/lib`, or download the latest version from Jakarta Commons I/O site.
- copy "lib/dwr.jar" in `[webapp_root]/WEB-INF/lib`, or download the latest version from Direct Web Remoting site
- copy the "javascript/scopes" directory ("/src/javascript/scopes if you are building from source) to [webapp-root]/scopes;
- configure DWR servlet by modifying your "web.xml" file:

```

<servlet>
  <display-name>DWR Servlet</display-name>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
     <param-name>debug</param-name>
     <param-value>true</param-value>
  </init-param>
</servlet>
<!-- ... -->
<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

- configure the `CompleteHttpScopesFilter` in front of your servlet, by modifying your "web.xml" file:

```
<filter>
  <filter-name>scopesFilter</filter-name>
<filter-class>net.sourceforge.scopes.filters.CompleteScopesHttpFilter</filter-class>
  <init-param>
  <param-name>manager-scope-names</param-name>
  <param-value>request,window,session,application</param-value>
  </init-param>
  <init-param>
  <param-name>manager-class-names</param-name>
  <param-value>
    net.sourceforge.scopes.managers.impl.RequestScopeManager,
    net.sourceforge.scopes.window.WindowScopeManager,
    net.sourceforge.scopes.managers.impl.SessionScopeManager,
    net.sourceforge.scopes.managers.impl.ApplicationScopeManager
  </param-value>
  </init-param>
  <init-param>
  <param-name>rewriter-node-visitor-classes</param-name>
<param-value>net.sourceforge.scopes.window.WindowNameNodeVisitor</param-value>
```

```
  </init-param>
  <init-param>
  <param-name>link-pattern</param-name>
  <param-value>*.do,windowTest,windowTestR</param-value><!-- Put the
URLs that will be rewritten -->
  </init-param>
  <init-param>
  <param-name>redirect-classes</param-name>
<param-value>net.sourceforge.scopes.window.WindowRedirectResponseRewriter</param-val
  </init-param>
</filter>
<filter-mapping>
  <filter-name>scopesFilter</filter-name>
  <servlet-name>myServletName</servlet-name>
</filter-mapping>
```

- Installation finished! For the use see the Quick start

4.1. Using "window" scope without code injection

You can use the window scope without the rewriter, to avoid code injection. This is the case if you are worried about performances, or if you are not using HTML.

First of all, remove the `rewriter-node-visitor-classes` parameter, or at least remove the value that references `net.sourceforge.scopes.window.WindowNameNodeVisitor`.

In your pages you need to put this code somewhere, such as in the `<head>` part (replace `${context}` with your web application context name, this case will be improved ASAP):

```
<script type='text/javascript'
src='${context}/dwr/interface/WindowManager.js'></script>
<script type='text/javascript' src='${context}/dwr/engine.js'></script>
<script type='text/javascript' src='${context}/dwr/util.js'></script>
<script type='text/javascript'
src='${context}/scopes/window/windowScope.js'></script>
```

In the `<body>` part of you page add the `onload` event handler this way:

```
<body onload="initWindowName();">
```

Into each `<form>` tag add the `onsubmit` event handler this way:

```
<form action="..." onsubmit="appendWindowNameToAction(this);">
```

Into each `<a>` (link) tag add the `onclick` event handler this way:

```
<a href="..." onclick="appendWindowNameToLink(this);">
```

Lastly, replace every `window.open` calls this way (the parameters are the same as in `window.open`):

```
openWindowWithWindowName(param1, param2,...)
```

5. How window scope works

The window scope needs more synchronization between the client and the server. For this reason, the client must inform the server on the window that made the requests.

Each window has a name, that acts as an identifier: in fact, if you try to open a window with the same name of another one, the request is made on the old window. Thus, there should be a way to send the window name to the server. This is done by the use of JavaScript that rewrites the URL by adding a cookie parameter (";window=NAME") that will never be put in a cookie. This way, it does not interfer with parameters.

The problem is that such a code is not present in an already made web application. Therefore, an HTML rewriter has been used, that injects JavaScript code in the generated HTML page through the use of a filter.

Each window attribute is stored in a `ContextImpl` object different for each window. Each context is stored in session scope with a different key for each window.

When a new page is open without the "window.open" JavaScript command, for example when you open the browser for the first time, of if you open a link in a new window/tab, the window itself has not a name. For this reason, some code is injected: this code checks if the window has a name, and if it has not any, it calls a server class (through the use of DWR) that generates a session-unique (i.e. it is unique within the same session) window name.

Last but not least, to preserve the window information when redirecting, a "Redirect Response Rewriter" is called, that appends ";window=NAME" to the original URL.