Lab 8: RichFaces Performance

In this lab, you will practice using some RichFaces performance techniques.

Objectives:

- To experiment with RichFaces performance issues

Part 1: Event Queues

In this part, you will experiment with using event queues to increase Ajax performance. We will provide you with a starter application that will send Ajax events based on mouse movement - you will use RichFaces techniques to throttle the Ajax requests to avoid "swamping" the server.

Steps:

_1. As you have done in earlier labs, import the **{Lab Installation Directory}/starters/lab08/lab08starter.zip** "Existing Project" archive into your workspace.

_2. Familiarize yourself with the provided code:

a. In the *Java Resources* folder, there are two Java files:

- **Box.java** represents a filled rectangle; it has a position, width, height and color.
- **BoxesBean.java** is a managed bean that maintains a list of boxes and has a *paint* method that draws the list to a bitmap using the Java Graphics2D object.

  This class also has a *randomize* method that creates a few boxes with random properties.

  There is also a *getHitBox* method that determines if a coordinate intersects a box. The coordinate, *hitX* and *hitY* are defined as properties in BoxesBean. Note also that this method has a *sleep* method that simulates heavy server load.

b. In the *WebContent* folder, there are several provided files:

- **boxes.xhtml** uses an *a4j:mediaOutput* component to display the bitmap drawn by the BoxesBean *paint* method.

  It also defines a command button that invokes the BoxesBean *randomize* method so the user can see different boxes.

  It also displays the results of calling the BoxesBean *getHitBox* method - when you are done with lab, this will show the properties of a box if the user moves the mouse cursor over it.
- **clearsession.jsp** is a simple JSP that invalidates the session and redirects to boxes.xhtml.
- **hits.js** is a JavaScript file that defines two functions, *getMouseX* and *getMouseY*. These functions return the coordinates of the mouse.
- **template.xhtml** is a standard Facelets template. Note that its *head* element contains a *script* references to hits.js. Since boxes.xhtml references this template, code in boxes.xhtml can call the two JavaScript functions defined in hits.js.

c. The **faces-config.xml** and **web.xml** are standard JSF and Web configuration files. Please examine both and ensure you understand them.

_3. Run *boxes.xhtml* on the server and test the provided functionality.

Note that as given, moving the mouse across the boxes has no effect. You will add that ability in a later step.

However, you can press the *Randomize* button and the component will redraw a new set of random boxes.

_4. Your first job is to add function so that as the user moves the mouse over the boxes, your program sends Ajax requests that include the current mouse coordinates. Follow these steps:

a. Open boxes.xhtml into the editor and find the close tag for the h:form element.

b. Immediately before the h:form close tag, define a Ajaxified JavaScript function using a RichFaces tag:

```
<a4j:jsFunction
   name="findboxJS"
   reRender="underbox">
   <a4j:actionparam name="x" assignTo="#{boxesBean.hitX}"/>
   <a4j:actionparam name="y" assignTo="#{boxesBean.hitY}"/>
</a4j:jsFunction>
```

This defines a JavaScript function named *findboxJS* that accepts two parameters, x and y. When the function runs, it will send an Ajax request that includes the parameter values; note how we assign the parameters to properties in the BoxesBean.

When the Ajax response comes back, we then re-render the *underbox* component, which displays information about any box under the x,y coordinates. Please re-examine BoxesBean.java and ensure you understand how this will work.

c. Configure the *a4j:mediaOutput* component so that mouse-move events call the *findboxJS* JavaScript function, passing the current mouse coordinates. Add the following attribute to the *a4j:mediaOutput* start tag:

```
onmousemove="findboxJS(getMouseX(event),getMouseY(event));"
```

Note that this invokes the two JavaScript functions you looked at in the hits.js file in an earlier step.

_5. Run boxes.xhtml on the server, then press the Randomize button until you see three non-overlapping boxes.

Then move the mouse over the bitmap. As you do so, look in the Eclipse Console and note that the getHitBox method is being called.

Move the mouse over one of the boxes, then stop moving the mouse. Patiently watch the Eclipse console until messages stop coming in. Note that there may be a several-second lag since the server cannot keep up with the numerous Ajax requests. The "Box Under Cursor" panel should (eventually) display characteristics of the box.

Note: The JavaScript code that returns the mouse coordinates is somewhat temperamental, so the "hit testing" might not be reliable. You may need to move the mouse to the center of the box before you get a "hit".

_6. The code as written so far works, but is not very efficient because the server is "swamped" with numerous Ajax requests. You can make the application more efficient and usable by throttling Ajax requests using RichFaces *event queues*.

Find the start tag for the *a4j:jsFunction* element, and add the following attributes to the start tag:

```
eventsQueue="myqueue"
ignoreDupResponses="true"
ajaxSingle="true"
requestDelay="50"
```

These attributes do the following:

- **eventsQueue** defines a queue for Ajax requests. RichFaces will thus queue up multiple outstanding Ajax requests and send them as a single, cumulative request.
- **ignoreDupResponses** causes the client-side Ajax JavaScript to "throw away" any response it receives if the queue is non-empty. The idea here is that the queued request will immediately change anything in the application that would've been modified by the ignored response.
- **ajaxSingle** is not queue-related, but speeds up processing on the server by skipping certain of the JSF phases.
- **requestDelay** tells the client-side Ajax JavaScript to wait 50 milliseconds before sending any Ajax request. So if more requests happen during that interval, they will be combined and sent as a single request.

_7. Try running boxes.html on the server again. Now it should be more responsive and less "laggy" and you should see fewer requests displayed in the Eclipse console.

_8. If you wish, experiment with the attributes you just added and see what difference they make. In some cases, it might be difficult to see much of a change.

Part 2: Configuring for Performance

In this part, you will experiment with various configuration settings that can improve RichFaces performance.

Steps:

_1. Add a Ajax button to your boxes application so you can experiment with performance configuration:

a. In BoxesBean.java, add a simple *submit* action method that returns a null String and simply prints a message to the console.

b. In boxes.xhtml, within the h:form element, define an Ajaxified command button that invokes the "submit" method:

```
<a4j:commandButton value="Test"
  style="position:absolute;left:420px;top:150px;width:100px;"
  action="#{boxesBean.submit}" />
```

c. Run boxes.xhtml on the server, then click the mouse on a blank area of the Web page and press Ctrl+Shift+L to bring up the Ajax logging window.

Click the Test button, then look in the Ajax log and note the following:

- In the *Full response content*, the response *head* element contains several links (references) to scripts and stylesheet files.

  In the Ajax log, starting immediately after the colon after *Full response content:*, highlight text until the close of the *html* element. Then copy that text into the clipboard.

Back in Eclipse, create an empty XML file named **before.xml** and paste from the clipboard into it. Then press Ctrl+Shift+F to reformat the text.

d. Close the Ajax log and all browser windows.

_2. In Eclipse, edit web.xml, and then add the following:

```
<context-param>
  <param-name>
    org.richfaces.LoadScriptStrategy
  </param-name>
  <param-value>ALL</param-value>
</context-param>
<context-param>
  <param-name>
    org.richfaces.LoadStyleStrategy
  </param-name>
  <param-value>ALL</param-value>
</context-param>
```

Then repeat the above step, copying and pasting to an XML file in Eclipse named **after.xml**.

Compare the two XML files and note how *after.xml* is much smaller - reducing the size of Ajax responses can have a significant affect on performance, especially if you have a high volume of Ajax requests: each response takes up less network bandwidth and the client-side Ajax JavaScript has less text to parse, making it faster.