MasterKey JSF

Tutorial

Building a Pazpar2 search interface in JavaServer Faces

Introduction

1. Purpose

This tutorial shows how to put together a search interface on top of MasterKey, simple yet complete with search box, results list, and pagination within about 25 lines of XHTML code.

The resulting XHTML file is shown in the last page.

2. Install the tutorial application

You can download the tutorial application from Index Data's ftp server at http://ftp.indexdata.com/pub/mkjsf/tutorial-app/. Deploy mysearchapp.war to you server and access the first page of the tutorial, for instance by http://localhost:8080/mysearchapp/mysearch-01.xhtml.

Subsequent tutorial pages are found in the directory /tutorial-files/ - i.e. http://localhost:8080/mysearchapp/tutorial-files/mysearch-02.xhtml

3. Tutorial

3.1. The empty MK2 JSF page

This is our empty start-up page. It contains the name spaces we will need as we proceed through the tutorial.

The page is found under the name `mysearch-01.xhtml` in the root of the tutorial web application.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<h:html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:pz2utils="http://java.sun.com/jsf/composite/pz2utils"
      xmlns:pz2widgets="http://java.sun.com/jsf/composite/pz2widgets">
<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>MySearch</title>
</h:head>
<h:body>
  <h2>Tutorial: MasterKey JSF</h2>
</h:body>
</h:html>
```

3.2. Adding a search form and including the polling mechanism

To prepare our search page for searching and polling the Pazpar2 Service Proxy, we will add three items:

- an authentication request as required for access to the Service Proxy
- a search form
- an Ajax polling mechanism for getting more results from Pazpar2

As we add functionality to the page it will go into this search form, which should eventually contain all our query and display logic.

The polling helper makes the browser keep asking for more updates from the server as the search proceeds in Pazpar2.

The polling mechanism is added by putting this tag in the page `<pz2utils:pz2watch id="pz2watch"/>` as shown in the box below.

We will set polling in debug mode for now. That will display the `activeclients` field, which the component uses to determine when to stop polling. `activeclients` is a count of pending search targets in Pazpar2. For more details, see the Pazpar2 documentation.[1]

Please also note the parameter `renderWhileActiveclients="".` Once you start adding search results to you page, you will add the tag ID's of those sections to this parameter, to instruct JSF to render those sections whenever new data comes in from Pazpar2.

The body of our page should look like this:

```
<h:body>
  <!-- This adds a convenience component that will poll pazpar2 until our searches
complete:
       Debug is set to true for now, so we can follow the polling.
-->
  <pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="" debug="true" />

  <h:form id="pz2form" prependId="false">
    <!-- Search and display logic will go here                              -->
    <!-- In five steps we will add a search box, a results list, a pager
         for browsing through the pages and a field to customize the page
         length, 18 lines or so of XHTML code.                              -->

  </h:form>
</h:body>
```

3.3. Adding search logic

We can now add search functions to the page. We need a field for the user to type in some query terms and a button to execute the search. For that we use a back-end object named `pzreq.` `pzreq` gives the page author access to all Pazpar2 commands and their parameters.

1 With debug on, you will see activeclients fields for the 'show' command and the 'record' command. Only the first one applies to this tutorial.

pzreq's counterpart is pzresp, another named object that provides access to all Pazpar2's responses on executed commands. We'll get to the responses shortly.

```
<h:form id="pz2form" prependId="false">

  <!-- Adding a field for the query, refer it to the bean property 'query' -->
  <h:inputText id="myquery" value="#{pzreq.search.query}" size="50"/>

  <!-- Adding a search button -->
  <h:commandButton id="button" value="Execute My Search">
     <f:ajax execute="myquery" render="${pz2.watchActiveclients}"/>
  </h:commandButton>
</h:form>
```

The input field will set the user's search terms as the query parameter of the search command (pzreq.search.query).

Clicking the search button will then do two things: Send changes of the input text field myquery to the server and trigger the polling by subsequently requesting that the special field ${pz2.watchActiveclients} be rendered.

If you enter a query and press the button, you should see a number for active clients – usually between 20 and 30 targets the first time around. The number should then be decreasing until it reaches zero. To retry the process, you would enter a new query. (Note: repeating the same query will have no effect since the back-end recognizes that it already has results ready for that query in memory.)

3.4. Showing results

We will now add a results list to the page. The results list will fetch the Pazpar2 show data from the back-end bean and loop the hits. While we are at it we will put sequence numbering in front of each record.

```
<h:panelGroup id="myshow">
   <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
   <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
      #{(pzresp.show.start)+loop.index+1}.<b>#{hit.title}</b>
           <i>#{hit.author}</i> #{hit.date}<br/>
            #{hit.description}<br/>
   </ui:repeat>
   </ui:repeat>
 </h:panelGroup>
```

Note the ID of this section – id="myshow".

This is the section that we want updated on each poll as described in 2.2. The polling tag should thus be modified like this: renderWhileActiveclients="myshow"

```
<pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow" debug="true" />
```

The body of the page should now look something like this:

```
<h:body>
                                           <!-- render new UI element when polling -->
  <pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow" debug="true" />

  <h:form id="pz2form" prependId="false">
    <h:inputText id="myquery" value="#{pzreq.search.query}" size="50"/>
    <h:commandButton id="button" value="Execute My Search">
       <f:ajax execute="myquery" render="${pz2.watchActiveclients}"/>
    </h:commandButton>
    <br/><br/>
    <h:panelGrid>
      <!-- Display the results in a new section named 'myshow'
           After each poll render this section by including it like shown above -->
      <h:panelGroup id="myshow">
        <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
           #{(pzresp.show.start)+loop.index+1}.<b>#{hit.title}</b>
                <i>#{hit.author}</i> #{hit.date}<br/>
                 #{hit.description}<br/>
        </ui:repeat>
      </h:panelGroup>
    </h:panelGrid>
  </h:form>
</h:body>
```

If this page is opened soon after the previous page in the tutorial, the previous search could still be in memory so results would come up without polling. Otherwise we just make a new search – or clear the browser cache to redo the old one.

3.5. Setting display options

In the previous example the page showed the first 20 results, which is the default. We will now add some very simple controls to change what sequence number to start with and how many results to show on the page.

These settings are display instructions, meaning they will not require a new search and thus no polling. We will add these two tags for setting the record to start with and the number of records to show:

```
    Start <h:inputText id="start" value="#{pzreq.show.start}">
            <f:ajax render="myshow"/>
          </h:inputText>
    Pagesize <h:inputText id="pagesize" value="#{pzreq.show.pageSize}">
                <f:ajax render="myshow start"/>
            </h:inputText>
```

Next, when we change the record sequence number to start the display with, we want the results list to be updated to reflect this. This is done by `render="myshow"`.

When we set the page size, we want that too, but – additionally – the back-end will always reset the start number back to the first record, when the page size is changed, and we should thus make sure to re-render the start field as well to reflect that.

The body of our page should now look something like this:

```
<h:body>
  <pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow" debug="true" />

  <h:form id="pz2form" prependId="false">
    <h:inputText id="myquery" value="#{pzreq.search.query}" size="50"/>
    <h:commandButton id="button" value="Execute My Search">
       <f:ajax execute="myquery" render="start ${pz2.watchActiveclients}"/>
    </h:commandButton>
    <br/><br/>
    <!-- Add rough navigation/paging in results -->
    <!-- 'start' specifies the first record to show -->
    <!-- 'pageSize' how many records to show -->
    Start <h:inputText id="start" value="#{pzreq.show.start}">
            <f:ajax render="myshow"/>
          </h:inputText>
    Pagesize <h:inputText id="pagesize" value="#{pzreq.show.pageSize}">
                <f:ajax render="myshow start"/>
               </h:inputText>
    <h:panelGrid>
      <h:panelGroup id="myshow">
        <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
           #{(pzresp.show.start)+loop.index+1}.<b>#{hit.title}</b>
                <i>#{hit.author}</i> #{hit.date}<br/>
                 #{hit.description}<br/>
        </ui:repeat>
      </h:panelGroup>
    </h:panelGrid>
  </h:form>
</h:body>
```

The sequence numbering that we display with `#{(pz2.show.start)+loop.index+1}` should now reveal if the pagination works.

3.6. Adding a slightly more sophisticated pager

The `start` and `pageSize` parameters are the basics of all the pagination you can do, but using them directly like this is a bit rudimentary. In the tutorial there is a pager that can be included in the page like this:

```
<h:panelGroup id="pager">
   <controls:pager renderOnChange=":pz2form:pager :pz2form:myshow"/>
</h:panelGroup>
```

The component will draw a list of page numbers – depending on the number of hits returned When a page number is clicked, the search results should be updated to show the selected page of course, but also the pager itself, to reflect that it is on a new page, thus: `renderOnChange=":pz2form:pager :pz2form:myshow"`.

We should also make sure the pager is updated on polling because the display of the pager will depend on the total number of results returned from Pazpar2. The polling tag should thus look like this:

```
<pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow pager"
debug="true"/>
```

The body of our page now looks like this:

```
<h:body>
  <pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow pager"
        debug="true" />
  <h:form id="pz2form" prependId="false">
    <h:inputText id="myquery" value="#{pzreq.search.query}" size="50"/>
    <h:commandButton id="button" value="Execute My Search">
       <f:ajax execute="myquery" render="myshow pager ${pz2.watchActiveclients}"/>
    </h:commandButton>
    <br/><br/>
    <h:panelGroup id="pager">
       <controls:pager renderOnChange=":pz2form:pager :pz2form:myshow"/>
    </h:panelGroup>
    <h:panelGrid>
      <h:panelGroup id="myshow">
        <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
           #{(pzresp.show.start)+loop.index+1}.<b>#{hit.title}</b>
                <i>#{hit.author}</i> #{hit.date}<br/>
                 #{hit.description}<br/>
        </ui:repeat>
      </h:panelGroup>
    </h:panelGrid>
  </h:form>
</h:body>
```

3.7. Apply a minimum of styling

There is much more to be done with this UI of course. Check out the demo at `http://jsfdemo.indexdata.com/` for more examples.

For now we will just remove the debugging of the polling, add a style sheet for minimal styling (with `<h:outputStylesheet library="css" name="styles.css"/>)` and put back the `pageSize` control so we can still set page size along with the pager. The full Faces page on next page concludes our tutorial.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<h:html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:pz2utils="http://java.sun.com/jsf/composite/pz2utils"
      xmlns:pz2widgets="http://java.sun.com/jsf/composite/pz2widgets">

<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>MySearch, pager and page size</title>
  <!-- Adds hard-coded authentication -->
  <h:outputText value="${pzreq.sp.auth.runWith2(
                             'action=login;username=guest;password=guest',';')}"
                rendered="${pz2.authenticationRequired and
                         not pzresp.sp.auth.isAuthenticationOk()}"
                style="display:none;"/>
  <!-- Adds a style sheet -->
  <h:outputStylesheet library="css" name="styles.css"/>
</h:head>
<h:body>
  <pz2utils:pz2watch id="pz2watch" renderWhileActiveclients="myshow pager"/>
  <h:form id="pz2form" prependId="false">
    <h:inputText id="myquery" value="#{pzreq.search.query}" size="50">
    </h:inputText>
    <h:commandButton id="button" value="Search">
       <f:ajax execute="myquery" render="myshow pager ${pz2.watchActiveclients}"/>
    </h:commandButton>
    <br/><br/>
    <h:panelGrid>
    <h:panelGrid columns="2">
     <h:panelGroup id="pager">
       <pz2widgets:pager renderOnChange=":pz2form:pager :pz2form:myshow"/>
     </h:panelGroup>
     <h:panelGroup>
       <!-- We might still want the page size option with the pager of course -->
       Page size <h:inputText id="pagesize" value="#{pzreq.show.pageSize}" size="2">
                   <f:ajax render="myshow pager"/>
                </h:inputText>
     </h:panelGroup>
     </h:panelGrid>
     <h:panelGroup id="myshow">
        <ui:repeat value="#{pzresp.show.hits}" var="hit" varStatus="loop">
           #{(pzresp.show.start)+loop.index+1}.<b>#{hit.title}</b>
                <i>#{hit.author}</i> #{hit.date}<br/>
                 #{hit.description}<br/>
        </ui:repeat>
     </h:panelGroup>
    </h:panelGrid>
  </h:form>
</h:body>
</h:html>
```