JavascriptWithJavaServerFaces

Is there a place for Javascript with JSF? I would say definitely yes.

One very common scenario is to introduce a confirmation dialog associated with an action (for example deleting a record, canceling an edit). Another common scenario is to perform client side validation thus saving a round trip to the server. An example is to check that the 'Password' and 'Confirm password' are equal before posting back to the server.

The article describes how to trigger client side javascript functions with the <h:commandLink> and the <h:commandButton> components.

<h:commandLink>

Associating a javascript with a commandLink is not difficult, however in order to do it successfully you need to understand how the <h:commandLink> component is rendered into HTML by JSF.

The example below illustrates how the <h:commandLink> is rendered in HTML:

```
<h:form id=”userForm”>
<h:commandLink id=”lnkDeleteUser” value=”delete” action=”#{userBean.deleteUser}”/>
</h:form>
```

```
<a href="#" onclick=
"clear_userForm();
document.forms['userForm'].elements['userForm:_link_hidden_'].value='userForm:lnkDeleteUser';
if (document.forms['userForm'].onsubmit){
      if (document.forms['userForm'].onsubmit())
document.forms['userForm'].submit();
} else {
      document.forms['userForm'].submit();
}
return false;
"
id="userForm:lnkDeleteUser">delete</a>
```

There are a few points to note:

- The <h:commandLink> is rendered as a hyperlink, <a href />
- The hyperlink itself is “#”; it is basically a dummy value.
- JSF generates a block of Javascript and it is tied to the ‘onclick’ event. Disregarding the details, it basically calls submit() which post the form to the server.
- Line 3 is of particular interest – the id of this particular component (“userForm::lnkDeleteUser”) is saved to a hidden field. This is how the JSF engine knows which particular component does the postback and to invoke at the server side actions appropriately.

Most JSF component allows us to inject javascript associated with various client side DHTML events like onclick, ondoubleclick, onfocus etc. With <h:commandLink>, since JSF is already generating Javascript associated with the onclick event, this is where we need to inject our own javascript codes.

The scrpt below illustrates how to inject a line of code to open a confirm dialog window, and the resulting HTML:

```
<h:form id=”userForm”>
    <h:commandLink id=”lnkDeleteUser” value=”delete”
            onclick=”if (!confirm('Are you sure you want to delete this record?')) return false”
            action=”#{userBean.deleteUser}”/>
</h:form>
```

```
<a href="#" onclick=
"if (!confirm('Are you sure you want to delete this record?'))
        return false;
clear_userForm();
document.forms['userForm'].elements['userForm:_link_hidden_'].value='userForm:lnkDeleteUser';
if (document.forms['userForm'].onsubmit){
      if (document.forms['userForm'].onsubmit())
document.forms['userForm'].submit();
} else {
      document.forms['userForm'].submit();
}
return false;
"
id="userForm::lnkDeleteUser">delete</a>
```

Another important point to note is that the javascript block should not return true under any circumstance. It is does so, the browser will proceed to perform <a href="#"> – which is redirecting the browser to the dummy "#" page.

<h:commandButton>

The command button is a little simpler. The script below illustrates how to inject a confirmation dialog with <h:commandButton> and how it is rendered in HTML.

```
<h:commandButton id="btnCancel" value="Cancel"
      onclick="if (!confirm('You will lose all changes made. Are you sure?')) return false"
/>
```

```
<input id="userForm:btnCancel" name="userForm:btnCancel"
      type="submit" value="Cancel"
onclick="
if (!confirm('You will lose all changes made. Are you sure?')) return false;
clear_userForm();
"/>
```

Here the commandButton is rendered as a HTML submit button. If the javascript block returns true then the form is submitted as usual. If it returns false then the form submission is aborted.

The key to successful javascripting with JSF is to understand what is being rendered. Some basic understanding of javascipt goes a long way as well. Also be extra careful with the syntax, as most IDE do not have any support for Javascript. If you made a syntax error - for example missing out a closing bracket, your codes will simply fail silently.

Using javascript and command links to submit a form from a control event

It is quite often that you may want to submit immediately from a control (combo box changes, radio button is selected, etc.). There is no easy way to get a specific action to execute when this happens. One way is via JavaScript and hidden command links.

JavaScript (tested on IE and FireFox):

```
function clickLink(linkId)
{
  var fireOnThis = document.getElementById(linkId)
  if (document.createEvent)
  {
    var evObj = document.createEvent('MouseEvents')
    evObj.initEvent( 'click', true, false )
    fireOnThis.dispatchEvent(evObj)
  }
  else if (document.createEventObject)
  {
    fireOnThis.fireEvent('onclick')
  }
}
```

Then add a hidden link to your page (use CSS to ensure it is not visible to the user):

```
<t:commandLink id="hiddenLink" forceId="true" style="display:none; visibility: hidden;" action="#{put your
action here}" actionListener="#{put action listener here}">
<!-- parameters, more action listeners, etc. -->
</t:commandLink>
```

You can see the use of "forceId". The reason for this is that we will be referencing this link using JavaScript and we need the ID and the client ID of a component is not easy to get from a JSF view.

Here is a check box control that will submit on change:

```
<t:selectBooleanCheckbox value="#{bean value goes here}" onclick="clickLink('hiddenLink');" />
```

Andrew Robinson
