Experiences With DynaFaces

By Roger Keays, 15 November 2006

AJAX is all the rage. When it first appeared on the scene, AJAX sent alarm bells ringing all over my office. You know why of course don't you? Yep, that's right - Javascript. I'm fairly certain the folks at Netscape had had way too much coffee when they decided that everything in life is actually just an array.

Anyway, recently there's been a lot of noise around JSF and AJAX being a perfect fit; about libraries that let you do AJAX without coding any Javascript and the such. While I'm not the biggest fan of Javascript, I can recognize the vast improvements in user experience that AJAX brings, so after some time I finally took the plunge.

I wanted to choose a library that just added AJAX functionality without all the cruft. Something simple, like being able to declare a region of the page that is updated using AJAX. DynaFaces [1], Ajax4jsf [2] and AjaxAnywhere [3] all offer components to do this but I chose to work with DynaFaces for a few reasons:

- It is the most lightweight.
- It's partial request processing make AJAX requests virtually equivalent to normal requests.
- As such, all the normal JSF concepts apply.
- It is led by a JSF EG member.

Unfortunately, DynaFaces is also the most immature.

Bugs in DynaFaces

The first thing I tried was a simple <ext:ajaxZone> to declare a region of the page which was updated using AJAX. It didn't work. I wasn't that fussed though - since this is early access software and I'm using Facelets, which probably hasn't been tested. Without investigating the problem too much I went straight on to try the javascript method of triggering AJAX requests. I'd need to use this anyway for what I had in mind.

A simple

```
<h:commandButton onclick="DynaFaces.fireAjaxTransaction(this, { render: 'panelG
                value="submit"/>
```

worked fine to redraw the panelGrid in my form, but that is where the caveats begin:

1. DynaFaces did not escape <![CDATA[and]]> markup which is generated by components for a partial request. Since <![CDATA[is used for the XML response's <markup> element, the result is invalid XML containing an (illegal) nested CDATA. [4]
2. The string ~ facelets.VIEW_STATE ~ appears in the output when rerendering forms because DynaFaces hijacks the StateManager and doesn't allow facelets to replace that key in AJAX requests. This means that no view state is sent to the server on subsequent form submissions and actions listeners are not invoked. [5]
3. Using the onclick attribute on a commandLink also doesn't invoke the actionListeners. It seems that the javascript is interfering with the setting of hidden request parameters required to determine which command* was pressed. [6]

They are the three serious blocker defects, but there are a couple of other niggling problems worth knowing about.

- DynaFaces keeps telling you that 'DynaFaces is already defined', until you uncomment that line in com_sun_faces_ajax.js.
- You can only rerender visible items, because the clientId is needed to replace the element. A simple workaround is to use style="display: none;" for the invisible item. This way it can still be refreshed with AJAX (and that display style removed).
- Using <ext:scripts/> in the body can cause some of the divs to flicker when they are redrawn in both Firefox and IE. The workaround is just to put that into the <head> section.
- The DynaFaces javascript cannot appear in the onsubmit form attribute, because submit() is called from scripts, which doesn't actually invoke to onsubmit event.

A Working Setup

Of the three critical defects mentioned above, the first has been fixed in the subversion repository, and a patch for the second is available on the issue tracker. The third, however, remains open. The workaround I am using is to use the form's submit method instead of the commandLink's onclick method to invoke the ajaxTransaction:

```
<script type="text/javascript">
    $('editForm').submit = function() {
       DynaFaces.fireAjaxTransaction(this, { render: 'editPanel' });
    }
</script>
```

If you need a conditional ajax submit, you can do something like this:

```
<script type="text/javascript">
```

```
    ajax=true;
    $('editForm')._submit = $('editForm').submit
    $('editForm').submit = function() {
        if (ajax == true) {
           DynaFaces.fireAjaxTransaction(this, { render: 'editPanel' });
        } else {
           this._submit();
        }
    }
</script>
```

and use the onclick method of the commandLink to set the ajax variable.

If you want to use DynaFaces now, my recommendation is to check it out from subversion and build it from source. This is quite easy, since the build uses maven. You should expect the occasional hiccup though, and if you're using facelets you'll also want to apply the patches attached to defect 18 [5].

Postmortem

When I started this process, I wanted to know if JSF and AJAX really are 'a perfect fit'. Ultimately, I think my conclusion is *yes*, with DynaFaces they are... but not yet. DynaFaces provides a partial request lifecycle that makes using AJAX virtually transparent to the developer, but it is still only early access so don't delete your Javascript bookmarks just yet.

P.S. What's with the name 'DynaFaces'? 'Avatar' sounds way cooler :)

References

[1] https://jsf-extensions.dev.java.net
[2] https://ajax4jsf.dev.java.net
[3] http://ajaxanywhere.sourceforge.net
[4] https://jsf-extensions.dev.java.net/issues/show_bug.cgi?id=13
[5] https://jsf-extensions.dev.java.net/issues/show_bug.cgi?id=18
[6] https://jsf-extensions.dev.java.net/issues/show_bug.cgi?id=15

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.