Ideas for JSF 2.0

By Roger Keays, 1 April 2007

So it looks like there is action stirring in the JSF camp, with a draft JSR for JSF 2.0 being released [1]. I know everybody complains about how slowly things move in the JCP, but to be honest, I hadn't expected this so soon. It looks like those guys at Sun have got a pretty clear idea of where they're going with this, but for what its worth, here is a list of things I'd like to see in JSF 2.0 that I've collected over the last year or so.

I've organised this habitually like I would a set of requirements for a particular job - that is, by category, or concern. For items that I think are especially important I've put in a few extra **s.

View technology

**** Master/client templating**

Wow. This one should have been a requirement for, like JSP 1.0. I've never written a webapp that didn't require a reusable master template. Tiles offers a JSP based solution to this problem, but I think Facelets' approach is much better. <ui:decorate> and <ui:param> are probably two of my most used features from Facelets. It just needs to be standardized.

**** JSF components as first class**

Having to write a JSP tag for each component, validator and converter is a real barrier for adoption - especially for newbies. I don't think it's just laziness either. Component development is hard enough as it is and all this extra work just introduces more potential failure points. Since I became a Facelets user and no longer have to do this, I'm much happier developing components, validators and converters.

**** Standardizing AJAX / partial request support**

For me, DynaFaces and ajax4jsf have only really proven themselves in theory and not necessarily in production. This is mostly because of tricky corner cases I keep managing to produce. When it does work, however, AJAX can make the UI consummate! I wouldn't have any problems with standardizing the DynaFaces partial request lifecycle, provided the spec gives enough information to solve any problems with invoking javascript for various use cases (e.g [2]).

The second part of this problem is providing an API or conventions for component developers so that they don't have to reinvent the wheel with each new library (and distribute these wheels either!).

Lifecycle

Separate build and render phases

This probably would have been how the JSF 1.0 EG would have liked it, but backwards compatability with JSP made a separate Build View phase impossible. Among the advantages of separating the build and render phases, I see the possibility of a <h:head> component, capable of collecting css and

js resources required by components nested in the tree [3]. It would also make for simpler debugging, and allow for my next idea...

Invoke lifecycle on first request

I'd be interested to see how many people would find it useful to be able to initiate the JSF lifecycle on an initial page request, and not just on a postback. For me, it solves several problems (see my blog on Fake postbacks with JSF + Facelets), including the ability to execute standard JSF actions on an initial page request or after a session timeout. This idea requires separable view build and render functionality.

Page lifecycle

Even being able to invoke the JSF lifecycle on an initial page request isn't going to satisfy all use cases. Shale's ViewController provides an example of a simple page lifecycle with potential for standardization.

Deferred ValueChangeListeners

A ValueChangeListener will be invoked if the UIComponent it is attached to validates correctly and the value has changed. The idea behind a deferred ValueChangeListener is that it is only invoked if the *entire form* validates. In that way, you can be sure that the Update Model phase is reached. This is useful, for example, if you want to make sure that the new value is persisted to the database. A deferred ValueChangeListener might even be invoked *after* the Update Model phase.

Context

**** Conversation scope**

This is pretty self expanatory and has been implemented several times by Seam, ADF Faces and others. I'm using a home grown alternative, but I'd be very interested in a standard, permanent solution.

FacesContext available to filters and other servlets

I can't see any convincing reason why the FacesContext shouldn't be available to filters and other servlets. It'd be convenient, and as far as I'm aware, shouldn't be impossible [4].

EL in Java

Okay, it's not the responsibility of the JSF EG, but which do you prefer?

```
String foo = (String) ((SomeBean) FacesContext.getCurrentInstance()
        .getExternalContext().getApplicationMap().get("bean")).getConfig()
        .get("my.config.variable");
```

or

```
String foo = ${bean.config['my.config.variable']}
```

${bean} as an lvalue

On the topic of EL, I'd also find it handy to be able to use ${bean} as an lvalue:

```
<f:setPropertyActionListener target="${bean}" value="${someBean.newValue}'
```

You could determine the type and scope if ${bean} was already resolvable or was a managed bean. If not... well, that needs to be considered.

Components

**** Reuse component state available from templates**

One of the most difficult hurdles in component development for me was understanding state saving. Especially because, a large portion of the time, all the state for your component is specified in your JSP /Facelets template. Not only is it hard to understand, but it also seems to have fairly bad performance implications. Server-side state saving wastes memory with state in abandoned sessions, and client-side state saving seems to me inefficient since it practically serializes the whole component tree.

I realize this is a fairly difficult problem, but it seems there are some ideas out there included statelessness where appropriate, and only storing state deltas [4].

**** Standard way of packaging, resolving and serving static resources**

This is another one that has been reinvented too many times. I don't think it's a difficult problem, it just needs a standard solution. This solution should probably also include a way of allowing components to add resources to the <head> (or equivalent) section of the output. That's a bit more complicated, but separate build/render phases would help (see above).

Miscellaneous component improvements

- Ability to specify <h:form> method attribute (GET or POST)
- Configurable default timezone for <f:convertDateTime/>
- Configurable default styles for <h:messages/>
- Make UISelectMany choose a converter-for-class for E from a List<E> genericized type
- Ability to define converters for annotated classes (e.g. @Entity)

Non-requirements

Here are some of the common suggestions that I personally wouldn't have a great use for in JSF 2.0:

- Backwards compatability with JSP(x).

- XML-free configuration.
- Flash scope. This would be obsoleted by a conversation scope, and may not always work where AJAX is in heavy use.
- Skinability / theming.
- Client side validation library.
- Calendar component.

That's not to day they aren't good ideas. I'm just not fussed about seeing them in the JSF 2.0 spec.

Comparison to the Draft JSR

I was pleased to read the draft JSR and discover that all the items aboved marked with stars are already proposed for JSF 2.0. My ideas about a Build View phase and other lifecycle features probably divert from the mainstream, but it seems that the problems they solve might be addressed in other ways.

In general, I'd say it looks great .

References

[1] https://javaserverfaces-spec-public.dev.java.net/proposals/JSF-2_0-draft.html
[2] https://jsf-extensions.dev.java.net/issues/show_bug.cgi?id=43
[3] http://www.nabble.com/A-simple-alternative-to-the-ExtensionFilter-tf3451557.html#a9628
[4] http://www.thoughtsabout.net/blog/archives/000033.html
[5] http://weblogs.java.net/blog/jhook/archive/2006/01/experiment_goin_1.html

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.