JSF is a diamond in the rough, you just need to make it shine

by Benny Bottema - Sunday, May 24, 2009

http://www.bennybottema.com/2009/05/24/jsf-is-a-diamond-in-the-rough-you-just-need-to-make-it-shine/

[RE: JSF – Still pretty much a steaming pile of donkey turd] – I was replying to Wille Faler's post about why JSF sucks, when the comment was getting too large, so I made it into a post on my own blog.

JSF is hard to learn yes, but in my opinion it doesn't suck as bad as Wille says. All the points Wille mentions can be solved by certain libraries or write-once reusable solutions (I've included my 'magical' combination of frameworks on the bottom). JSF is a diamond in the rough, you just need to make it shine.

Contents

The devil is in the details

There are a couple of pitfalls you need to know about with JSF. It's all really a matter of having read a good book about JSF that points them out, or having learned them the hard way. Luckily I've had some guidance by experienced colleagues of mine; I'm guessing that's the missing factor in Wille's situation.

Referring to the pitfalls you need to know about, I'm talking about the common beginner's problems like using JSTL c:tags that cause havoc when combined with those of jsf and facelets. I'm talking about the Ajax4Jsf details as Wille mentioned: once you know what to use it's easy. Those 48 attributes? most of them are inherited and never used and you can forget about them, and that's the learning curve. Once you know the right ones, it's using exactly those, over and over again. And things like how you can incorporate Spring security and stuff like that. Another pitfall that comes to mind is how action="" work quirky with navigation rules when using the array notation (action="controller['action']"). And ofcourse there's the issue of parameterized JSF validators, which needs some understanding. All pitfalls that can baffle a beginner or intermediate.

The thing with JSF is there are a number of little things that are hard to solve, but once you're past that it *does* make you much more productive. You just have to have a solid base to start from each project. The pitfalls learning curve is definitely a downside of JSF, but JSF 2.0 is in the making to solve all these issues.

Bad performance or beginner's mistake?

Then there is this performance issue Wille mentions, where I think he missed the target: JSF does not have to be slow at all. It really depends on how you treat your backing beans. JSF has the weird tendency to call setters/getters multiple times, which seems utterly pointless (it's related to resolving valuebindings in nested components). I've made the mistake before to directly retrieve values from a controller/service in my views instead of a static model (backing bean): this is what makes things slow. JSF in itself isn't that slow for a webapp: yes there are many redundant calls but we're not dealing with a realtime simulation application, webapps do just fine. On a sidenote, it also depends on the implementation… JSF is a reference specification, while there are various implementations of the framework (IBM's, Sun's etc.)

SEO friendly JSF?

Ahh, but for the JSF Search Engine Friendliness. That's solved too, somewhat. There's is a framework called PrettyFaces which allows you to dynamically generate/interpret pretty GET urls with bean values. It's a great framework that solves a common problem, but there still exists the problem that existing jsf/facelets/richfaces components are not generating SEO HTML code. That's really the fault of the component makers, not JSF, though JSF definitely isn't innocent with its POST obsession: PrettyFaces fixes at least that.

Common sense: use frameworks to avoid most pitfalls

Once you've got those kind of things down and use a good framework combo, JSF is straightforward. You won't really need to know about the various phases and you can be very productive if you've set up the jsf/facelets components right. I've worked with JSF intensively for about a year now in various projects with various implementations and I hardly ever need to use phaselisteners and that sort of stuff. The rare cases where you do need them, you can define these things ones and reuse them in other projects. With this approach I've created a template project with some standard libraries, phaselisteners, security set up, facelets components etc. and I'm up and running in no time.

I haven't sanitized or otherwise cleaned up the template project so I won't put that up for the time being, but at least I can tell you what frameworks are in it by default.

> The magical combination that works for me and my colleagues:

- JSF (MVC)
- Facelets (Templating, use of XHTML)
- Richfaces (component suite)
- Richfaces A4J (Ajax support)

- Tomahawk (only for file uploads with the Extensions Filter)
- PrettyFaces (for indexable GET urls)

Now that I've mentioned these, you should check out JBoss Seam as well, which basically is supposed to do everything the above libraries do combined, plus a couple of things more like solving the conversational scope issue. I haven't worked with this framework myself though so I can't speak from experience.

Final note

Although I agree JSF has some annoying pitfalls, shortcomings and a high learning curve, it's been out there for enough time to pass for supplement frameworks to pop up from under the ground to solve most of these problems.

With the right combination, most classic counter arguments have become obsolete. Maybe it's time for critics to take JSF as is: incomplete but ultimately making you more productive if combined with the right frameworks.
