Parameterized JSF Facelets Validators

by Benny Bottema - Saturday, July 26, 2008

http://www.bennybottema.com/2008/07/26/parameterized-jsf-facelets-validators/

You'd be surprised how hard it is to find good information about how you're supposed to setup validators that you can parameterize with attributes in your tags; most about validators is very basic and very JSF-only. Or you won't be surprised at all, because you searched until loss of sanity like me and found this blog. Whatever, here you can read what I found out.

So at first I made the error in not realizing validators under Facelets should be defined in their own facelets tag library. I hadn't really worked myself into the the theory behind Facelets; I just picked it up as I went… as I do with most things, but there you have it. Turns out Facelets actually makes things very easy concerning validators, were it not for one sneaky little detail:

"**parameterizable validators are stateful validators**"

We'll come back to it, but validators generally lose state after the page has been built by Facelets and you have to manually restore that state (including the previously set parameters), by implementing the StateHolder interface.

Oracle has a good article on StateHolder Validators, though I only found it after I learned what to look for.

I resorted to a rather crude catch'em all solution, that tucks away the boilerplate code involved.

Contents

Creating a parameterizable validator

Having validators with configurable properties involves the following basic steps:

- *1. create a JSF validator with the properties you wish to set in your tags*

- *2. register the validator with your faces-config as you normally would in JSF*
- *3. create a Facelets taglibrary file and register it with your application*
- *4. add an entry to this tablib file that defines a validator and points to the faces-config one*
- *5. that's it, no properties need to be explicitly defined, Facelets will take care of that*

We'll add one more step to it to make the validators actually work:

- *1.5. modify the validator with a StateHolder solution*

1. Create a JSF Validator

So for this example I'm going to take a simplified version of the age validator I made earlier in Streamline your JSF validation framework.

AgeValidator.java

So this validator already has some properties for min and max age we can use. In the previously post article they didn't work properly though, we'll fix that today.

2. Register the validator with JSF in faces-config

So before we can use the validator in Facelets, JSF needs to recognize it. Here's the piece of xml that does that in faces-config.xml:

```
<validator>
 <validator-id>quintor.validator.AgeValidator</validator-id>
 <validator-class>nl.quintor.commons.validator.AgeValidator</validator-
class>
</validator>
```

So you may have noticed I didn't really use a normal validator id. That's because I don't want to confuse them with normal validators for use in JSF only and to illustrate their significance in wiring them to Facelets. Instead I'm using a sort of grouping id by using a shortened version of their full package (for converters it would be *quintor.converter.SomeConverter*).

3. Create a Facelets tag library file and register it with the application

To be able to use the validators in Facelets, and thereby making use of Facelets parameterizing capabilities, we need to use yet another xml file that Facelets needs.

Create a file in the META-INF webapp folder; I'll call mine 'quintor-validators.taglib.xml'. I picked that up from another project that used this convention. Next the taglib file needs to be registered with the

web.xml:

```
<context-param>
  <param-name>facelets.LIBRARIES</param-name>
 <param-value>
  /WEB-INF/quintor-validators.taglib.xml
 </param-value>
</context-param>
```

4. Add entry that uses the faces-config validator

Here’s the validator definition inside quintor-validators.taglib.xml:

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC "-//Sun Microsystems, Inc.//DTD Facele
t Taglib 1.0//EN" "facelet-taglib_1_0.dtd">
<facelet-taglib>
 <namespace>http://www.quintor.nl/jsf/validator</namespace>
 <tag>
  <tag-name>validateAge</tag-name>
  <validator>
   <validator-id>quintor.validator.AgeValidator</validator-id>
  </validator>
 </tag>
</facelet-taglib>
```

1.5. Finally let’s focus on the ‘state problem’

Currently, the age validator is ready to go, if it weren’t for the fact it forgets any value set in the tag. For instance, take the following tag example:

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:ui="http://java.sun.com/jsf/facelets"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:qnt-validator="http://www.quintor.nl/jsf/validator">

<ui:composition>
 <ui:define name="body">

 <h:inputText>
```

```
  <qnt-validator:validateAge min="18"/>
  <ui:insert/>
 </h:inputText>

 </ui:define>
</ui:composition>
</html>
```

Here the minimum age of 18 is set for validation. If we were to debug the validator, we'd notice the setter is being called when the page is being build up. If we then submit the form, the validator is recreated, but the setter is not being called again thus the validator reverts to its default values!

How then, can we keep the values set by the attributes? The answer lies in the StateHolder interface that comes with JSF, or rather why you *need* StateHolder. The problem lies in the fact that JSF keeps everything page scoped by default and stores all components in the Response object (which would explain why the setter is being called during page-build). Through the use of the StateHolder interface, JSF is able to carry a component's state across requests and responses.

An alternative to StateHolder would be to make the validator *serializable*, as the MyFaces wiki reveals on JSF State Management, but that's a rather implicit solution. I'd prefer the explicit version by implementing StateHolder. Instead of having each and every validator implement StateHolder, I chose for a more 'catch all' solution. This solution involves an abstract base class that stores the complete state of a given component (validator or converter) and restores this.

Here's the class and GenericUtils which harbors some boilerplate code:

AbstractStateholder.java
GenericUtils.java

So what happens is that we just ignore transient and let JSF deal with that. But when saving/restoring state we just crudely take Apache's BeanUtils and create a clone of the current validator instance and take over all of its properties, which will now need *getter* methods as well. The copied state is limited by what properties on the validator has getters/setters defined for them. When JSF is asking the validator to restore its previously stored state, we just slam the cloned validator's properties on the current one.

Here's the modified AgeValidator now: AgeValidator.java modified

The new version only has some getter methods added so that the abstract stateholder class can retrieve the values for state-saving. Now, the validators are being parameterized by Facelets (common conversions done automagically) and the attributes on the tags are being remembered on the validator to the point of actual validation.

Nice, huh.
