WhyNotContexts

NOTE: This proposal only applies to older versions of Tapestry.

Mikaël Cluseau

Introduction

The goal of this document is to provide a solution to the problem highlighted by the following sample, without changing a line of it :

Problem.page

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE page-specification PUBLIC
  "-//Apache Software Foundation//Tapestry
Specification 3.0//EN"
  "http://jakarta.apache.org/tapestry/dtd/Tapestry_3_0.dtd">
<!-- generated by Spindle, http://spindle.sourceforge.net -->
<page-specification class="org.apache.tapestry.html.BasePage">

    <property-specification name="fieldNames"
        type="java.util.List"
        initial-value="{'f1','f2','f3'}"/>

    <property-specification name="fieldName"
type="java.lang.String"/>

    <property-specification name="value"
type="java.lang.String"/>

    <bean name="required" class="org.apache.tapestry.valid.StringValidator">
        <set-property name="required" expression="true"/>
    </bean>

    <bean name="delegate"
class="org.apache.tapestry.valid.ValidationDelegate"/>

</page-specification>
```

- Problem.html*

```
<html jwcid="@Shell" title="Home">
<body jwcid="@Body">
<form jwcid="@Form" action="#" delegate="ognl:beans.delegate">
    <table border="0">
        <tr jwcid="@Foreach" source="ognl:fieldNames"
value="ognl:fieldName"
            element="tr">
            <td style="text-align: right;">
                <span jwcid="label@FieldLabel"
field="ognl:components.field">
                     Field name
                </span>:
            </td>
            <td>
                <input jwcid="field@ValidField"
                    displayName="ognl:fieldName"
                    value="ognl:value"
                    validator="ognl:beans.required"/>
            </td>
        </tr>
    </table>
    <input type="submit" jwcid="@Submit"/>
</form>
</body>
</html>
```

The problem

The problem is that is you enter something in the first field (`f1`) and not in `f2` and `f3`, you will see that `f1` and `f3` field labels are in red, and `f2` and `f3` are decorated.

Analyze the generated component hierarchy :

- Shell > Body > Form > Foreach
    - label$1 (label "f1")
    - field$1(field "f1")
    - label$2 (label "f2")
    - field$2 (field "f2")
    - label$3 (label "f3")
    - field$3 (field "f3")

I've volontary began at "label$1" and "field$1" to avoid naming ambiguity.

Obviously, this is because FieldLabel gets its field by calling `getComponent("field")`, which returns the last component called field that have been reached: after the rewind phase, it is `field$3`, which is empty, so `label$1` sets itself in red; `label$2` gets `field$1` so doesn't sets itself in red; `label$3` gets `field$2` so sets itself in red. Even this last case is false at the semantic level.

A non-intuitive solution

Knowing the problem, a solution can be found on the Howard's blog (see Howard's blog). He creates a components named `Defer` ; this component renders a block in a buffer then, on its ouput, its body and the buffer. This way, the block is actually rendered before the field and then the getComponent ("field") returns the expected component. You'll see that it works if you download my sample project ([TODO: URL]), but it is not very intuitive even at the code level: we need to put the label's `td` into the `Defer` and the field's `td` into the `Block` because otherwise the rendering is not done as it appears in the template (since `Defer` renders it after its body).

Contexts

Principles

A context as I understand it here is a set of attribute-value elements and of subcontexts. It also specify value search in a nearest-match way: the search is recursive, until it finds the attribute or exhaust the root-context.

Why use them ?

Mainly because the humain brain work with contexts, highlighting parts of its mind as needed. Concequently, the developper expects that the references to field are bound to the corresponding label ; I mean, the one within the same loop.

Using contexts to store the right reference to a component is a way to make templates more intuitive to the developper because the underlying model will correspond to the template's intuitive view.

Why is it resolving our problem ?

It will solve our problem if the Foreach creates new contexts for each iteration because, then, instances of field will be bound to the current loop's context, and thus the getComponent("field") call will return this instance even if it is after the label in the template (and in the render sequence).

The tree would be :

- {context: root}
  - Shell > Body > Form > Foreach
  - {context: Foreach@loop$1}
    - label ($1)
    - field ($1)
  - {context: Foreach@loop$2}
    - label ($2)
    - field ($2)
  - {context: Foreach@loop$3}
    - label ($3)
    - field ($3)

I put "$X" in parenthesis because I want to highlight the fact that now we don't event care of the instance we're accessing, we already know that it is the right one. The naming-wall of the context allows (structural) coherency between the template's component naming and the instance at the tree level.

How to do it ?

We could do it quite easily if we register each component's instance within its container with its unique name and with its name suffixed by "$X" in the page. The main problem I see is that we may need to create the document's tree before processing the parameter binding.

Howard, I may try to do it whenever I catch some time and after Tapestry 3.1 has gone to the high gear...

HowardLewisShip: Good luck; I don't see this working. You are looking for a more Swing like appoach, were you create new components as needed to deal with repeats. Tapestry's model is to render the same components multiple times. There are a large number of reasons, related to efficiency, pooling and scalability, for keeping the structure of a page stable. Tapestry's one-pass approach is very important for dealing with side effects (such as avoiding multiple database queries) and complexities such as loops-within-loops and the use of Block/RenderBlock to dynamically choose what gets rendered.