JSF c:forEach vs ui:repeat

By Roger Keays, 7 June 2007

This is probably one of the most frequently asked questions on the JSF mailing list. **Why doesn't my c:forEach tag work correctly?** Unfortunately, there are may ways to misuse the JSTL tags available in JSF, so the answer isn't always simple. Here is an explanation of the differences between c:forEach and ui:repeat, along with some examples which will hopefully save you some headaches.

TagHandlers vs Components

The most important thing to understand about the JSTL tags in JSF is that they **do not represent components** and never become a part of the component tree once the view has been built. Rather, they are tags which are actually **responsible for building the tree** in the first place. Once they have done their job they expire, are no more, cease to be, etc etc.

Here is a table of the semantics of several common tags. I just discovered, reading the Facelets code, that validators and converters are classified separately. I had always thought they were just tag handlers, but I imagine they behave in much the same way.

TagHandlers	Components	Other
c:forEach c:choose c:set c:if f:facet f:actionListener f:valueChangeListener ui:include ui:decorate ui:composition any custom tag file	ui:repeat ui:fragment ui:component f:view f:verbatim f:selectItems h:inputText h:datatable any custom UIComponent	f:validator f:converter

One of the problems here is that there is no naming convention to indicate which tags correspond to which constructs. You've just got to know, or find out.

When is the view built?

Now that you understand that tag handlers are only effective when the tree is built, the next logical question should be well, when is tree built?

The short answer is that a new view is built for every request which is not a postback. During a postback, the view is reconstructed from saved state. Quite confusing, and not very obvious I know, but there you have it.

Common laments

The most common pitfalls are either with the JSF lifecycle, EL evaluation or combining tag handlers with components.

My c:if always evaluates to false

```
<h:dataTable values="${numbers}" var="number">
  <h:column>
    <c:if test="${number > 5}">
      <h:outputText value="${number}"/>
    </c:if>
  </h:column>
</h:datatable>
```

Yes, the c:if is always evaluating to false! But it is only ever evaluated once - when the tree is built. The h:outputText component never makes it into the tree. Solution: replace the c:if with:

```
<ui:fragment rendered="${number > 5}"> ... </ui:fragment>
```

You could also use the rendered attribute on the h:outputText component in this example.

My ui:include fails inside ui:repeat

```
<ui:repeat value="#{bean.items}" var="item">
 <ui:include src="#{item.src}"/>
</ui:repeat>
```

The EL for the ui:include is evaluated when the view is *built* and is invalid since it relies on a variable only made available by the ui:repeat during *rendering*. Use c:forEach in this case.

My recursive tag never stops

myTag.xhtml:

```
<ul>
  <ui:repeat value="${item.children} var="child">
    <li><eg:myTag item="${child}"/></li>
```

```
  </ui:repeat>
</ul>
```

The stop condition in this recursion is supposed to be when you run out of children. The problem is that the custom eg:myTag is just a tag handler, like a special version of ui:include. When the view is built, the ui:repeat has no influence on the building process and can't stop the recursion. Use c:forEach here instead of ui:repeat. Or better still, convert your tag file to a real UIComponent.

You might also recognise that the ${child} EL expression is meaningless during build time in this example, unless you use c:foreach.

My list doesn't change size after deleting or adding an item

```
<h:form>
  <c:forEach items="${list}" var="item">
    <h:outputText value="${item.name}"/><br/>
  </c:forEach>
  <h:commandButton value="Create new item" action="..."/>
  <h:commandButton value="Delete an item" action="..."/>
</h:form>
```

When your view was built you only had, say, 5 items. If you post back to this view and add or delete an item, your view still has 5 h:outputText components in it since it was restored from saved state. In this simple case, you should use ui:repeat and your tree will always contain one h:ouputText component which is iterated over with differing values of ${item}.

If you rely on using c:forEach to dynamically include different form components you could run into difficulty. Always try to think about what the resulting tree looks like and remember it doesn't change on a postback.

Suggestions for the future

Probably the best relief to this problem would be to come up with a better syntax or naming convention to distinguish between tag handlers and components. I imagine you could also improve compilation performance if you did this.

Secondly, we need better terminology. I've used the terms *tag handler* and *component* in this blog which isn't too bad. The Facelets' FAQ [1] uses the terms *build-time tags* and *render-time tags* which is a bit misleading because render-time tags (components) are involved in all phases of the JSF lifecycle, not just the Render View phase.

Whatever happens, tag handlers are very useful (would you use facelets without ui:decorate?) so let's not get rid of them.

References

[1] http://wiki.java.net/.../FaceletsFAQ#Why_doesn_t_my_c_if_ui_repeat_ui

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.