ListView and other repeaters

Table of contents

A repeater is Wicket's way of doing loops and allows the user to add a series of items or components where the number of items is not available at design-time.

The most common repeater is `wicket.ListView`, but there are also a number of components in the wicket-extensions' `repeaters` package.

FAQ

Q1. I added behavior (for example a `SimpleAttributeModifier`) to the ListView but I do not see its results in the output. Why?
Q2. I called `ListView#setRenderBodyOnly(true)` but the enclosing tags are still generated. How come?
Q3. My validation messages are cleared. What did I do?

A1 and A2: When you want to change the default generated markup it is important to realise that the ListView instances does not correspond to any markup but the generated ListItems do. This means that methods like `Component#setRenderBodyOnly(boolean)` and `Component#add(IBehavior)` should be invoked on the ListItem that is given in the `ListView#populateItem(ListItem)` method.

A3: See section below on repeaters and form components.

A quick introduction

Code:

```
List list = Arrays.asList(new String[] { "a", "b", "c" });
ListView listview = new ListView("listview", list) {
    protected void populateItem(ListItem item) {
        item.add(new Label("label", item.getModel()));
    }
};
add(listview);
```

Markup:

```
<span wicket:id="listview">
   this label is: <span wicket:id="label">label</span><br/>
</span>
```

Will result in the following rendered to the browser:

```
<span wicket:id="listview">
   this label is: <span wicket:id="label">a</span><br/>
</span><span wicket:id="listview">
   this label is: <span wicket:id="label">b</span><br/>
</span><span wicket:id="listview">
   this label is: <span wicket:id="label">c</span><br/>
</span>
```

Using form components in a repeater

⚠ Due to the dynamic nature of such repeaters as ListView, RefreshingView, and DataView you may want to use a less dynamic repeater such as RepeatingView to build your form. However, if you do need a really dynamic form where rows come and go on the fly then see below...

You can put anything in the repeater, including FormComponents. You can bind values to the form components through IModels just as if they were in a regular form.

There are however a few provisions you have to take care of when using a repeater in the form. Usually repeaters clear out their items at the beginning of every request, when inside a form this is usually undesirable because you want to preserve the old items because you want them to keep their state as opposed to being recreated fresh.

For example, if you use ListView, you should call `ListView.setReuseItems(true)` inside the form so that it preserves old items instead of always creating new ones everytime.

ListView Table Example

Lets take a simple example, we want to show a table of users:

```
<table>
  <tr wicket:id="listview">
    <td><span wicket:id="firstname"></span></td>
    <td><span wicket:id="lastname"></span></td>
  </tr>
</table>
```

What we want is everything within `<tr>...</tr>` to repeat once for every user, so we attach ListView to the `tr` tag.

What ListView does is make its immediate children believe that their markup is that of the ListView itself, so each immediate child believes that it is attached to the `<tr wicket:id="listview">` tag.

We need to represent these immediate ListView children with a container component, because each of these children may contain multiple children of its own, in our case the firstname/lastname labels. This is what the `ListItem` is, its a simple WebMarkupContainer created for you so you don't have to do it everytime yourself.

So we will have a ListItem container created for every list item in the list that feeds the listview. but this is not enough, as each of these list items must contain the components that represent data within it (the firstname, lastname labels that we want to add).

This is where `#populateItem(ListItem)` comes in. It is called once for every listitem created and allows you to add components to it, so the listview render process looks like this:

Before each render:

1. clear all children
2. for every list item in the list:
3. create a ListItem webmarkupcontainer
4. add created ListItem as a child
5. call populateItem with the created ListItem container to allow users to populate it

During render:

1. render all immediate children with own markup

In this example you just write:

```
add(new ListView("listview", userList) {
    protected void populateItem(ListItem item) {
        User user = (User) item.getModelObject();
        item.add(new Label("firstname", user.getFirstname()));
        item.add(new Label("lastname", user.getLastname()));
    }
});
```

The variable `userList` is initialized as follows:

```
List userList = Arrays.asList(
    new User[] {
        new User("FirstA", "LastA"),
        new User("FirstB", "LastB"),
        new User("FirstC", "LastC")
      });
```

where class `User` is defined as a simple java bean:

```
class User {
    String _first, _last;

    public User(String _first, String _last) {
        this._first = _first;
        this._last = _last;
    }

    public String getFirstname() { return _first; }
    public String getLastname() { return _last; }
}
```

Accessing the cell/row index:

```
protected void populateItem(ListItem item) {
        // cell index
        item.getIndex();

        // row index
        ((Item)item.getParent().getParent()).getIndex();
}
```

Optimizing the example with PropertyListView

When you are ok with using reflection, you can optimize the example by using PropertyListView. This works as follows:

```
add(new PropertyListView("listview", userList) {
    protected void populateItem(ListItem item) {
        item.add(new Label("firstname"));
        item.add(new Label("lastname"));
    }
});
```

This works because the model of each `ListItem` is wrapped in a `CompoundPropertyModel`.

Added by Gwyn 11:26, 8 May 2006 (BST) based on a post of Igor Vaynberg. Edited by Erik van Oosten to add the FAQ and PropertyListView example, and remove the wiki conversion mess.