Tree Tags

Table of contents

1. Introduction

The following topic explains the tree tags and classes and how they are used to create and render trees. A tree is rendered in an HTML page based upon an object representation of the tree. NetUI defines a set of classes which create the tree structure which is rendered. The object representation may be created either through NetUI JSP tags found in a JSP or may be created programmatically in a page flow or shared flow. This means that there are parallel representations of a tree. In a JSP, a set of JSP tags represent the tree. This representation is then transformed into a tree data structure defined by a set of tree classes.

2. Simple Example

This section presents a sample of the most basic tree. The tree is created in a JSP and displays a simple tree on the page. The tree itself has a root node with three children. The root may be expanded and collapsed. Any of the tree nodes may be selected.

2.1. Simple Example Code

simpleTree.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
    <head>
        <title>SimpleTree</title>
        <netui:base/>
    </head>
    <netui:body>
        <netui:tree dataSource="pageFlow.simpleTree"
selectionAction="postback" tagId="tree">
            <netui:treeItem expanded="true">
                <netui:treeLabel>0</netui:treeLabel>
                <netui:treeItem>
                    <netui:treeLabel>0.0</netui:treeLabel>
                </netui:treeItem>
                <netui:treeItem>0.1</netui:treeItem>
                <netui:treeItem>0.2</netui:treeItem>
            </netui:treeItem>
        </netui:tree>
    </netui:body>
</netui:html>
```

The `<netui:tree>` tag is the JSP tag that adds a tree to the page. It is responsible for rendering the tree in the generated HTML page. In this example, the contents of the tree itself are also defined in the `<netui:tree>` tag by using the `<netui:treeItem>` tags and

nesting them in a tree structure. In this simple case, the tree has a root (0) with three children (0.0, 0.1, and 0.2).

There are three required attributes on the `<netui:tree>` tag, `dataSource`, `tagId` and `selectionAction`. The `dataSource` is used to bind to a `TreeElement` based data structure representing the tree to be displayed. The `selectionAction` is the action that will be called when a node is selected in the tree. In some cases, it may also be called when a node in the tree is expanded or collapsed. In addition, the `tagId` attribute is also required. If `runAtClient` is `true` then you must also specify the name of the tree by setting the `tagId`.

Note: In the example above, the leaf nodes are defined in two manners. The first child (0.0) uses the `<netui:treeLabel>` to set the nodes label. The next two children (0.1 and 0.2) are defined with the label as the body of the `<netui:treeItem>`. The `<netui:treeItem>` supports setting the label value from the body of the `<netui:treeItem>` if it is a leaf in the tree. You are required to use the `<netui:treeLabel>` for all interior nodes or nodes with children. In other words, `<netui:treeItem>` does not support mixed content; meaning that interior nodes must use `<netui:treeLabel>` to set the label value and all text inside the body is ignored.

Controller.jpf

```
package simpleTree;

import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;
import org.apache.beehive.netui.tags.tree.TreeElement;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="simpleTree.jsp"),
        @Jpf.SimpleAction(name="postback", path="simpleTree.jsp")
    }
)
public class Controller
    extends PageFlowController
{
    private TreeElement _simpleTree;

    public TreeElement getSimpleTree() {
        return _simpleTree;
    }

    public void setSimpleTree(TreeElement _simpleTree) {
        this._simpleTree = _simpleTree;
    }
}
```

This very simple Page Flow controller supports displaying a tree. There is a single property "simpleTree" which holds a reference to a `TreeElement`. This `TreeElement` represents the root of the underlying tree object structure which is rendered by the `<netui:tree>` tag. There are two actions defined, the standard `begin` action and the `postback` action. The `postback` action is called when a selection or expansion link is selected in the rendered tree.

2.2. Simple Sample Lifecycle

This section describes the basic tree lifecycle using the SimpleTree example above. The following figure represents the basic lifecycle of a tree being rendered by a `<netui:tree>` tag.

Tree Tag Lifecycle

The `dataSource` attribute is a required attribute on all trees. It binds to an instance of a `TreeElement` (defined in `org.apache.beehive.netui.tags.tree`). All trees are represented as tree data structure with a single root. The `dataSource` attribute binds to this root.

When the `<netui:tree>` tag begins processing it checks to see if the variable bound to by the `dataSource` is equal to null. If it is, then the tree will process it's body content to create the tree data structure. If the `dataSource` is not null, then that tree's data structure is used to render content. If you want to programmatically (or dynamically) create a tree, you may create the tree structure before the bound variable is accessed. Typically this would be done in the `onCreate` method of a shared flow or page flow.

In the simpleTree sample, the first time the page is displayed the body of the tree tag is processed because the page flow's property `simpleTree` is null. This creates the initial tree data structure which is then rendered. When the page is requested again, for example when a tree element is selected, the tree data structure created on the first request continues to be used and the body of the `<netui:tree>` tag is ignored.

Note: A very common development task is to iteratively develop the `<netui:tree>`'s body content. In order for any changes to be reflected when the tree is rendered, the variable bound to by `dataSource` must be null. It is common to add an action to the page flow that will reset the value to null and call that from a link on a page. If variable is not null, changes to the JSP will not be reflected in the rendered tree.

3. Tag and Classes

This section describes the primary JSP tags and how they relate to the classes which define

the underlying data structure representing the tree. All of the tree features are available both in the tree tags as well as the underlying tree classes (described below).

3.1. Tag to Class Mapping

There are a number of JSP tags that allow creation of tree through JSPs. These tags build the underlying data structures representing the tree. This section describes the mapping between the JSP tags and the actual classes that represent the tree.

Note: in many cases, this document describes setting attributes on the tree JSP tags to enable features. In reality, the attributes are passed through to the `TreeElement` class which usually has a corresponding property. Other tags map their values to properties of the `TreeElement`. If you are programmatically creating a tree by building the tree hierarchy using `TreeElement`s, you directly set properties on the tree classes.

The following list describes the mapping of the tree JSP tags to underlying tree classes:

- **`Tree`** -- The `<netui:tree>` tag doesn't create a `TreeElement`. It binds to a `TreeElement` representing the root of the tree data structure. The `<netui:tree>` tag does create the initial `TreeRenderState` object representing how the tree is rendered.
- **`TreeItem`** -- The `<netui:treeItem>` tag will create a `TreeElement`. If the `<netui:treeItem>` is the root of a tree, then the `TreeRootElement` will be created.
- **`TreeLabel`** -- The `<netui:treeLabel>` tag sets the value of the label which is stored as a property of a `TreeElement`. Using this tag is required for non-leaf nodes. For leaf nodes the body content of the `<netui:treeItem>` will be used as the label value as long as that body does not contain other JSP tags (mixed content).
- **`TreeContent`** -- The `<netui:treeContent>` tag sets the value of the content for a `TreeElement`. The content is a property of the `TreeElement`.
- **`TreeProperyOverride`** -- The `<netui:treePropertyOverride>` tag is used to override various attributes on the tree such as the selection action and images. This tag will create an `InheritableState` object and set it on the `TreeElement`.
- **`TreeHtmlAttribute`** -- The `<netui:treeHtmlAttribute>` tag is used to set additional attributes on the HTML generated when rendering the node. This tag will create a `TreeHtmlAttributeInfo` class that is set on the `TreeElement`.

3.2. ITreeRootElement

In the SimpleTree example above, we described the tree data structure as being a hierarchy of `TreeElement` nodes. Many advanced features, including runAtClient, of the tree require the root of the tree to implement the interface `ITreeRootElement`. The class

`TreeRootElement` extends `TreeElement` and implements `ITreeRootElement`, providing a default implementation. In the SimpleTree example, when the body of the `<netui:tree>` is processed, the root `<netui:treeItem>` is created as a `TreeRootElement` and all other `<netui:treeItem>`'s are created as `TreeElements`.

The following features require the root element in a tree to implement `ITreeRootElement`:

- **`runAtClient`** -- Allows the tree to be expanded and collapsed on the client without round trips to the server.
- **Root Images** -- Allows setting different expand and collapse images on the root node of the tree.

The following additional state is tracked by the root element:

- **Selection** -- Direct access to the currently selected tree element.
- **Tree State** -- Access to the InheritableState and TreeRenderState defined on the tree (explained below).
- **Images** -- Allows different expand and collapse images to be set on the root supporting the Root Images feature.

4. Tree Features

This section describes the basic features of the NetUI Tree. The SimpleTree example introduces the basic mechanics for creating a tree in a page flow. A tree is output into the HTML page as a hierarchy of `TreeElements`. The SimpleTree example renders the following:

Tree Tag Display

The root of the tree supports expanding and collapsing. The children of a node appear at the same level. Trees appear commonly in applications such as file system explorers and are good at representing limited hierarchical data sets.

4.1. runAtClient

The `<netui:tree>` tag has an attribute `runAtClient` which when set to `true` will enabled expanding and collapsing the tree on the client without server round trips. When runAtClient is on, the tree will be completely rendered into the generated HTML. Client side JavaScript will then collapse and expand nodes when the user interacts with the tree. The following image describes the interactions between the server and client.

Flow of the tree when runAtClient is true

runAtClient uses `XmlHttpRequest` to update the underlying state on the server as the user interacts with the tree on the client. This mode requires JavaScript and XmlHttpRequest support in the client browser. This mode of operation is commonly referred to as AJAX (Asynchronous JavaScript and XML). It minimizes the amount of information sent between the client and server when the user is exploring the tree itself.

In the diagram above, when the tree is rendered, all of the nodes will be rendered into the HTML document generated. JavaScript on the client will then process the tree when the HTML document is loaded. The JavaScript will turn off display of tree nodes which are collapsed so that the tree appears in the expected state. As the user interacts with the tree by expanding and/or collapsing nodes, JavaScript will continue to turn on and off the display of tree nodes (and their children). In order to update the state of the tree stored on the server, the client also use XmlHttpRequest to send messages to the server indicating the nodes that are being expanded and collapsed. The next full server request will display the tree properly because the internal state has been updated as the user interacted with the tree.

4.2. expandOnServer

When a tree has the `runAtClient` attribute set, then individual elements can indicate that they need to be expanded on the server by setting the `expandOnServer` attribute on the `<netui:treeItem>` tag. When `expandOnServer` is enabled, if the node is in a collapsed state, the node itself will be rendered in the generated HTML, but all children nodes will not. When the user expands the node, an XmlHttpRequest is made to the server and the children (and possibly their children) will be rendered into HTML and sent back to the client. JavaScript will update the DOM and cause the children to be displayed. Once the children are received, all further expand and collapse operations happen on the client.

runAtClient and `expandOnServer` can be used together to optimize the amount of tree state rendered into the initial request and then to minimize the amount of state transferred when the user is exploring the tree. It is very common for people to drill into one or two areas of a tree after searching the top level nodes. To optimize for this type of browsing, render out the top few levels of a tree and then create a layer of children that set `expandOnServer` to true. The top few layers will be initially rendered and when a user goes deep into one, the server provides the branch asynchronously when requested.

4.3. TreeElement Rendered Contents

This section describes the markup written out to represent a tree node in the rendered HTML document. The basic Markup looks like this:

[Tree Markup] [Expand/Collapse Icon] [Anchor - [Icon][Label]] [Content]

- **Tree Markup** [

lineJoin.gif

- lineJoin.gif,

lastLineJoin.gif

- lastLineJoin.gif,

verticalLine.gif

-verticalLine.gif, spacer.gif] -- There are four images that represent the "structure" of the tree. These are used to create the visual hiearchical representation of the tree.

- **Expand/Collapse Icon** [

nodeCollapsed.gif

- nodeCollapsed.gif,

lastNodeCollapsed.gif

- lastNodeCollapsed.gif

rootCollapsed.gif

- rootCollapsed.gif,

nodeExpanded.gif

- nodeExpanded.gif,

lastNodeExpanded.gif

- lastNodeExpanded.gif

rootExpanded.gif

- rootExpanded.gif] -- There are six images that represent the expand and collapse links on an interior node. The root images are only available if the root of the tree implements `ITreeRootElement`.

- **Icon** [

folder.gif

- folder.gif,] -- This Icon and the Label represent the node in the tree. Either act as a selectable link that will call the selection action.
- **Label** -- The label is a text item representing the node. This is a property of the `TreeElement`. This is a selectable link that will call the selection action.
- **Content** -- This is an optional text item that may appear after the label. It is not selectable.

The tree supports setting a default location where the images are picked from within a WebApp. All of the images are found by default in the `resources\beehive\version1\images` directory. It is possible to change both the default location for finding the images in addition to the images themselves by explicitly setting the name of the image on the Tree.

4.4. Using a Custom TreeRenderer Implementation

The HTML markup for the tree is handled by the TreeRenderer class. By default, TreeRenderer handles tree rendering across the web application, unless another rendering class is specified.

You can override the rendering behavior of the default TreeRenderer class with a custom renderer class. A custom TreeRenderer class is especially useful for precise control of whitespace, line breaks, and image placement in the rendered tree.

To override the default TreeRenderer class:

1. extend the TreeRenderer class and override any of the formatting methods that are appropriate to your purposes
2. configure NetUI to use your extended class to render the tree

An example custom TreeRenderer class appears below. This class overrides the method renderConnectionImageSuffix() so that a new line is not added after the <img> element for the connetcting expand/collapse image and renderSelectionLinkPrefix() so that no white space indentation is placed before the anchor used to select a node. Also, the methods renderItemIconPrefix() and renderItemIconSuffix() are overridden to wrap a around the <img> element for the node icon. A might be used to incorporate CSS styles or a call to a JavaScript routine.

```
package mytree.renderer;

import org.apache.beehive.netui.tags.rendering.AbstractRenderAppender;
import org.apache.beehive.netui.tags.tree.TreeElement;
import org.apache.beehive.netui.tags.tree.TreeRenderer;
```

```
public class MyTreeRenderer extends TreeRenderer
{
   protected void renderConnectionImageSuffix(AbstractRenderAppender
writer,
                                              TreeElement node)
   {
   }

   protected void renderSelectionLinkPrefix(AbstractRenderAppender writer,
                                            TreeElement node)
   {
   }

   protected void renderItemIconPrefix(AbstractRenderAppender writer,
                                       TreeElement node)
   {
       writer.append("<span ID=\"myItemIcon\" style=\"cursor:pointer;\"");
       writer.append(" onClick=\"doSomething()\">");
   }

   protected void renderItemIconSuffix(AbstractRenderAppender writer,
                                       TreeElement node)
   {
       writer.append("</span>");
   }

    // more overridden methods...
}
```

To configure NetUI to use your custom TreeRenderer, edit the <tree-renderer-class> element of the beehive-netui-config.xml file to refer to your custom class:

```
<tree-renderer-class>mytree.renderer.MyTreeRenderer</tree-renderer-class>
```