Constructing reusable Web components using JavaScript objects

Nikolaj Cholakov

Abstract: *The efficiency of the Web applications development process can be increased significantly with the usage of reusable components, working not only on the server, but also on the client. In this article a clientside TreeView component is presented, intended for use in JSP pages as well as in "pure" DHTML pages. The realization is based on the JavaScript language's object-oriented capabilities, and supports tree-like presentation of textual and graphical data with possibilities for expanding/collapsing and element selection with clientside event handling.*

Key words: *Web programming, reusable components, Java Script , Java, JSP*

INTRODUCTION

The development of Web applications becomes much more effective and fast when the developer can rely on a wide range of reusable components. These components must be specialized, diverse, and easy to use and must have a common interface. It is also an advantage if they are platform independent.

One of the most important questions arising here is where should these components be used: on the server or on the client? If the Web application consists of dynamic pages only, obviously this should be a set of serverside components. But this will be very limiting – they could not be used for clientside scripting. On the other side these components should not be entirely clientside oriented, because then their use in dynamic pages will not be enough effective. A good decision here will be the creation of modular, platform independent serverside components which can get their content from different sources – databases, files or network connections, but at the same time their interface part can be used from clientside scripts.

There is another important problem concerning the management of the components. Often they are interactive, and when the user operates with the interface of the component, it changes its look, contents or behavior. To do this, the component has to handle the events fired from the user actions, and this handling can be done on the client or on the server, and more flexibility here can be achieved if both variants are realized.

LAYOUT

The Java Server Pages (JSP) specification provides a good possibility for the construction of such components with the JSP custom tag libraries. A component can consist of one or several JSP tags with the necessary set of attributes, forming the contents and the properties of the component. The rendering of the component on the user's page however should not be done directly. A better way is to use clientside scripts to construct the resulting HTML page, because these scripts could be used independently and thus a pure clientside version of the component can be made. JavaScript is a good choice for this purpose – simple, powerful and object-oriented.

Structuring information in a tree-like manner is a very common approach since it is natural and easy to deal with for both users and developers. Most of the environments for Graphical User Interface (GUI) applications include some form of a TreeView component, but there are difficulties for the web developers, because HTML lacks direct support for tree structures. As an implementation of the shortly described approach a Web component of type TreeView will be presented. It offers an easy way to construct fully functional classical tree structures in Web pages.

The following functionality is expected from a TreeView element:

- Presentation of textual and graphic data within a clearly defined and properly displayed tree structure;
- Expansion and contraction of tree nodes triggered by the end-user;

- Support for tree element selection, firing an event holding some kind of data;
- Handling of toggling and selection events;
- Different look for expanded, collapsed and selected elements.

The first thing to do is to construct the data structure of the TreeView component. It comprises a list of tree nodes on its top level. Each tree node in turn is composed of other tree nodes and tree leaves. The latter cannot contain other nodes. There is no top-level, root node by default. If there is only one node on the top level, that node in effect will be a root node. The data contents of every node and leaf are arbitrary, but typically it will be some text and/or images.

This article will not discuss particularly issues related to the construction of the data in the tree. As mentioned above, the data can be taken from different sources on the server, most frequently – a database. JSP custom tags give a good means to do this. The main tag starts the construction. Using the attributes of the main tag, the programmer can specify parameters concerning the whole component:

- name - the name of the component. There can be multiple TreeViews on a single Web page, and they must have different names;
- manageOn – determines where events triggered from the end-user action will be handled: on the server or on the client;
- selectionMode – allows the programmer to determine whether the selection feature will be allowed for nodes, leaves or both;
- onSelect – the name of the function (serverside or clientside) which contains the selection event handler;
- allowReselect – if true, allows multiple consecutive selection events on a single element;
- model – represents Java object, containing some kind of data model of the entire tree – for example a javax.swing.tree.TreeModel object. This is one of the possible ways to specify the data contents of the tree. The other way is to use separate JSP tags for each node and leaf; then the data for an element is in the body of the corresponding tag. If a model object is specified, the whole TreeView component consists of the tree tag only; otherwise the tree structure is specified through nested tags – one for the tree, and individual tags for each node and leaf.

In both cases a special render class renders the tree. This class constructs the HTML page containing the tree by creating several JavaScript objects. These objects contain all necessary fields and methods to cover all functionality of the tree. The main object presenting the whole tree is called Tree; its structure is shown in Table 1. Each tree element is presented with a TreeItem object (Table 2). Two helper objects are also defined – TreeItemChild and TreeItemIndent, with very simple structure. TreeItemChild represents a child of a tree element and contains two fields: the id number of the child and its type.

Table 1. Tree object elements

Element	Description
Fields	
name	The name of this tree
items	An array containing all elements of this tree, presented as TreeItem objects
lineHeight	The tree elements height in pixels
selected	The id of the currently selected element
selectAction	The name of the function handling the selection event
allowReselect	Boolean value indicating whether the reselection of an

	element is allowed
allowMultiple	Boolean value indicating whether multiple selection of tree elements is allowed
multipleSelection	An array containing id numbers of all currently selected elements
Methods	
addItem( )	Adds a tree item to the items array
render( )	Renders the entire structure of the tree
rearrange( )	Rearranges the visible structure of the tree (needed for some browsers only)
refresh( )	Redraws the visible structure of the tree after toggle operation
refreshSubTree( )	Redraws the visible structure of a sub tree
refreshNode( )	Redraws the contents of a node
scrollWindow( )	Scrolls the browser window to a new position
select( )	Performs element selection
changeImage( )	Changes the current image of a tree element

Table 2. TreeItem object elements

Element	Description
Fields	
id	The id number of this tree element
type	The type of the element: node or leaf
parent	The id number of the parent element
content	The content of the element in normal state
selContent	The content of the element in selected state
expCotent	The content of the element in expanded normal state
selExpContent	The content of the element in expanded selected state
image	The image of the element if any
toggleImage	The image used to toggle this element
expandByCaption	Boolean value indicating whether the element can be toggled by its caption
status	The current status of the element: expanded / collapsed
visible	Boolean value indicating whether the element is currently visible
children	An array containing all children of this element presented as TreeItemChild objects
indents	An array containing all indents of this element, presented as TreeItemIndent objects
Methods	
addChild( )	Adds a child to the children array
addIndent( )	Adds an indent to the indents array

TreeItemIndent represents an indent image for an element and contains only the name of the indent image.

The constructor function of the Tree object expects initial values for the following fields: name, lineHeight, allowReselect, allowMultiple. The "items" array of the Tree object contains all top-level elements of the tree as TreeItem objects. After the creation of every

top-level element object it is added to the "items" array using the addItem() function. The meanings of the other fields are obvious.

The constructor function of the TreeItem object expects initial values for the following fields: id, parent, type, content, selContent, expContent, selExpContent, image, toggleImage, expandByCaption, and status. Thus the content of this element for its four possible states is specified. Typically a tree element contains an image, specified with the "image" field followed by text data, but this is not obligatory: for example the data can contain other images. The "toggleImage" field specifies the image situated before every element and used to expand and collapse its contents. Typically this is an icon containing plus or minus signs, depending on the current state (See figure 1). The "status" field holds the current state of an element: expanded or collapsed. The initial state of a node can be specified with the "status" parameter of the constructor function.

The "children" array of every node contains all direct children of this element, presented as TreeItemChild objects. The array is constructed using the addChild( ) function. Of course leaves do not contain any children.

The exact display of the tree depends upon the defined style, but usually, child nodes and leaves appear indented to the right of parent nodes. The "indents" array contains all indent images, situated before each element and forming its indent. Typically two indent images are used: blank space or vertical line, connecting all direct children of an element (See figure 1). The "indents" array is constructed using the addIndent( ) function.

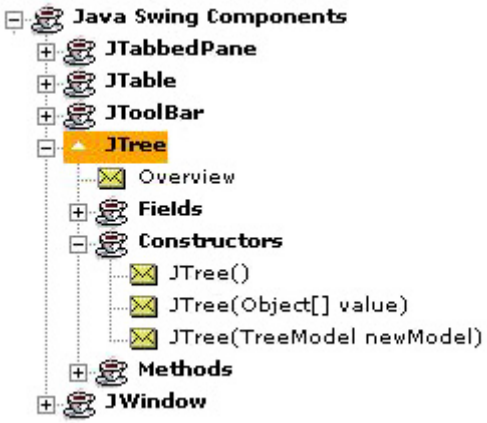

Figure 1. A sample TreeView component

Once the structure of the tree is created, the render( ) and refresh( ) functions of the Tree object has to be called to visualize the tree. The structure is drawn sequentially top to bottom and left to right where child nodes appear on the right and under parent nodes. But the visible part of the tree will change after each expand or collapse operation. That is why each tree element is situated in a positioned HTML box, depending of the client's browser. For DOM supporting browsers the DIV container is used, but for Netscape 4.x for example, the LAYER element is required.

A tree element can have totally different content in each of the four possible states, and only one of these contents can be visible at a time. When the user expands collapses or selects an element, the state of this element changes and the corresponding content becomes visible. To make this possible, the render( ) function places each content in a separate HTML container. Then for showing or hiding a container, the style properties of this content are manipulated: for DOM supporting browses – the "display" property, for others – "visibility" and "position".

The render( ) function constructs and writes out the whole HTML structure of the tree. Then the refresh( ) function determines all initially visible elements and makes all

necessary style changes. Some of the older browsers require absolute positioning of the elements, and the rearrange( ) function is intended for that purpose.

Every expanding or collapsing operation changes the visible structure of the tree. Therefore the refresh( ) function has to be called to update the tree. But to avoid needless operations only the affected part of the tree is refreshed using the helper functions refreshSubTree( ) and refreshNode( ). Also when the browser window content changes, some smart window scrolling helps the user to see better what happens on the page. This is done (if necessary) by the scrollWindow( ) function after each operation.

The TreeView component also supports the notion of node and/or leaf selection. When an element is selected, its content may change and an event may be triggered. This is determined by the select( ) function. For example if en element is already selected, reselecting is not allowed, and the user tries to select the element again, this action is simply ignored. If an event is triggered, it can be handled on the client or on the server. In both cases a string containing arbitrary data is associated with this element, and passed to the event handler.

To fully define the look of the tree, the programmer must specify how a node looks like, how a leaf looks like, what images to use to draw the tree structure – images are needed for lines, for the expanded and collapsed states of a node, and for spacing. All this can be done using different style templates on the server. Finally all the information is passed to the JavaScript objects. If nothing is specified, the look of the tree is very simple – just the content of the elements with the default font, color and background for the page.

The following code fragment demonstrates how Tree and TreeItem objects can be used to construct and render the tree from Figure1:

```
swing = new Tree('swing',16,'true','false','false');
…
swing.addItem('d1', 'd1.4', 'node',
   '<table  cellspacing=0 cellpadding=0 border=0>
      <tr>
         <td nowrap><img src="images/java.gif"/><img src="images/space.gif"></td>
        <td nowrap class="f10"><b><a href="javascript:showData('d1.4',d4,'true', 'swing')" >
            JTree</a></b></td>
      </tr>
   </table>',
   '<table  cellspacing=0 cellpadding=0 border=0 bgcolor="#e2e2e2">
      <tr>
         <td nowrap> <img src="images/java.gif"/><img src="images/space.gif"></td>
         <td nowrap class="f10"><b><a href="javascript:showData('d1.4',d4,'true', 'swing')" >
             JTree</a></b></td>
      </tr>
   </table>',
   '<table  cellspacing=0 cellpadding=0 border=0>
      <tr>
         <td nowrap><img src="images/java.gif"/><img src="images/space.gif"></td>
         <td nowrap class="f10"><b><a href="javascript:showData('d1.4',d4,'true', 'swing')" >
             JTree</a></b></td>
      </tr>
   </table>',
   '<table  cellspacing=0 cellpadding=0 border=0 bgcolor="orange">
      <tr>
         <td nowrap><img src="images/up.gif"/><img src="images/space.gif"></td>
         <td nowrap class="f10"><b><a href="javascript:showData('d1.4',d4,'true', 'swing')" >
             JTree</a></b></td>
      </tr>
   </table>', 'images/minus_middle.gif', 'images/plus_middle.gif','true','false');

swing.items['d1.4'].addIndent('space');
```

```
…
swing.render('images/line.gif',16,16,'images/space.gif',5,10);
swing.refresh();
```

Because the code is too big to be presented here fully, only the construction of the tree called "swing" and the node with caption "JTree" are shown. First of all the Tree object is created, with the given name and element height 16 pixels. Then all the elements of the tree are added to the structure one by one. The four HTML tables represent the four states of the node. They do not differ much: only by the background color; but there are no limitations and the four states can be completely different. After the addition of every element to its father element, the indents of the element are assigned. In this case the "Jtree" element has only one indent and it is a space, but the "Fields" element for example has two indents: a space and a line.

At the end the whole tree is rendered and only elements with currently expanded parents remain visible. The picture on figure 1 does not show the initial state of the tree. The "JTree" node has been expanded and then selected, and because of that its background color is different.

CONCLUSIONS AND FUTURE WORK

There are many popular tools and languages for Web programming both on the client and on the server side. They provide great functionality and diversity of tools. HTML and its extensions provide a powerful means for interface design, but they still lack the uniform integrated user experience that traditional desktop applications enjoy.

Reusable components, if properly designed and constructed, may bring Web development to the familiar, long-time proven programming style of desktop GUI programming - developers can work with controls having properties and responding to events. To reach this goal, the features of the server-side and client-side programming must be integrated into a common framework, so that the development of Web applications becomes easy, fast and much more effective.

REFERENCES

1. Eckel, B. Thinking in JAVA. Prentice Hall, New Jersey, 2001.
2. Ladd, E., J. O'Donnel. Using HTML4, XML and Java. QUE, Indianapolis, 1998

ABOUT THE AUTHOR

Nikolaj Cholakov, PhD, Department of Information Technologies, "St.st. Cyril and Methodius" university of Veliko Turnovo, Phone: +359 62 649831, E-mail: n.cholakov@uni-vt.bg