jQuery - Creating an Expandable Tree

Jakob Jenkov
Last update: 2015-01-26

You can use jQuery to create a graphical tree structure where the nodes in the tree can be collapsed and expanded. This tutorial will show a simple example of how such a tree structure can be implemented with jQuery.

To create a tree which can be collapsed and expanded we will use a **`ul` HTML element**. A `ul` element is an unordered list. Such a list can have nested lists recursively, so this element is already capable of displaying a tree structure.

The ul Tree Structure

Here is a `ul` element example that shows a `ul` as a tree:

```
<ul id="tree">
  <li>1
    <ul>
      <li>1.1
        <ul>
          <li>1.1.1</li>
          <li>1.1.2</li>
        </ul>
      </li>
      <li>1.2
        <ul>
          <li>1.2.1</li>
          <li>1.2.2</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

As you can see, the root `ul` element has nested li elements, which again have nested `ul` elements. This is how the tree is formed.

Tree CSS Styles

A `ul` list naturally displays bullet points next to each list item (`li` element). In order to remove the bullet points and adjust the indentation, we need to add a few CSS styles to the page. If you are not familiar with CSS, I have a free, thorough **CSS tutorial**.

Here is how the tree CSS classes are defined:

```
<style>
ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
}

li {
  padding-left: 0.5em;
}
</style>
```

We also need to define three custom CSS classes used by the generated tree. Here they are:

```
<style>
.handle {
  background: transparent url(/images/spacer.png);
  background-repeat: no-repeat;
  background-position: center bottom;
  display: block;
  float: left;
  width: 10px;
  height: 11px;
}

.collapsed {
  background: transparent url(/images/plus-black.png);
  background-repeat: no-repeat;
  background-position: center bottom;
  cursor: pointer;
}

.expanded {
  background: transparent url(/images/minus-black.png);
  background-repeat: no-repeat;
  background-position: center bottom;
  cursor: pointer;
}
</style>
```

The `handle` CSS class is a base style used for all expand / collapse handles. The `.collapsed` and `.expanded` classes are used by the handled for collapsed and expanded nodes in the tree. You will see how they are used once you see the code that generates the tree.

Tree jQuery Code

Finally, here is the jQuery code that turns the `ul` element into an expandable tree:

```
<script>
$(document).ready(function() {
  jQuery("#tree ul").hide();

  jQuery("#tree li").each(function() {
    var handleSpan = jQuery("<span></span>");
    handleSpan.addClass("handle");
    handleSpan.prependTo(this);

    if(jQuery(this).has("ul").size() > 0) {
      handleSpan.addClass("collapsed");
      handleSpan.click(function() {
        var clicked = jQuery(this);
        clicked.toggleClass("collapsed expanded");
        clicked.siblings("ul").toggle();
      });
    }
  });
}
</script>
```

First the code selects all nested `ul` elements inside the element with the id "tree" and hides them. This is done so that nested nodes in the tree start out collapsed. Only the top nodes of the tree are visible.

Second, the code selects all li elements inside the element with the id "tree" and iterates them.

For each li element, a span element is inserted inside it. This span element is the expand / collapse handle. By default it has the handle class set on it. If the li element also has an `ul` element as a child element, the class collapsed is added to the inserted span element, and a click listener is attached.

The has() function returns the subset of selected elements that has elements matching the given selector string. If the li elements has a `ul` element inside it, has("ul") will return a set containing the li element itself. If the li element does not have a `ul` element inside it, the set returned will be empty, and size() will return 0. Hence the check if(jQuery(this).has("ul").size() > 0) ,

which returns true if the li element has a nested ul element, and false if not.

The click listener function added to the handle span toggles the CSS classes "collapsed" and "expanded", and toggles visibility of the following ul element. Since the inserted span elements starts with the CSS class "collapsed", then toggling "collapsed" and "expanded" in the same statement will effectively switch between the two classes. The class already found will be removed (e.g. "collapsed"), and the class not found will be added (e.g. "expanded").

A Slightly Shorter Version

The above code could be made a bit shorter, by taking advantage of the function call chaining in jQuery. Most function calls that do not return a new object, return the object the function was called on. Hence, the above code could be written like this:

```
<script>
$(document).ready(function() {
  jQuery("#tree ul").hide();

  jQuery("#tree li").each(function() {
    var handleSpan = jQuery("<span></span>")
      .addClass("handle").prependTo(this);

    if(jQuery(this).has("ul").size() > 0) {
      handleSpan.addClass("collapsed").click(function() {
        jQuery(this).toggleClass("collapsed expanded")
          .siblings("ul").toggle();
      });
    }
  });
}
</script>
```

Furthermore, in a real code editor, two of the above chained function call lines could have been written in one line.

An Alternative Implementation

There are other ways to implements an expandable tree. Here is yet another way:

```
<script>
$(document).ready(function() {
  jQuery("#tree ul").hide();

  jQuery("#tree li").prepend("<span class='handle'></span>");

  jQuery("#tree li:has(ul)")
    .children(":first-child").addClass("collapsed")
```

```
    .children( ":first-child" ).addClass( "collapsed" )
    .click(function(){
      jQuery(this).toggleClass("collapsed expanded")
        .siblings("ul").toggle();
    });
}
</script>
```

First this implementation first hides all `ul` elements within the element with the id tree.

Second, the code selects all li elements within the element with id tree, and inserts (prepends) an empty span element into each selected li element. Notice how the class "handle" is set inside the span HTML code, rather than using jQuerys CSS functions. Sometimes it is easier to insert attributes and classes directly in the HTML. To do the same using jQuery I would have had to write this:

```
jQuery("#tree li").prepend("<span></span>")
  .children(":first-child").addClass("handle");
```

Finally the code selects the li elements (inside the element with id tree) that has a `ul` elements inside them, selects the first children of these li elements (the handle span elements), adds the "collapsed" class to them, and adds a click listener function. Quite simple, right?

As you can see, there are many ways to implement the same functionality using jQuery. Which implementation is preferable is a matter of personal taste.

And that is pretty much it! Of course, you may want more advanced features, like drawing of vertical lines between nodes of the same level, images that match the node type (e.g. file or folder), dynamic loading of nodes from the server for large trees etc. Rather than implementing all these features yourself, you should take a look at some of the existing tree implementations, for instance the jsTree mentioned in the beginning of this chapter. They will most likely save you a lot of time. This example merely served as an example of the use of jQuery.

Next: jQuery Critique

Jakob Jenkov