Displaying and sorting/paging tabular data using the JQuery tablesorter plugin, and query objects

Posted At : 12 February 2010 15:28 | Posted By : Shaun McCran
Related Categories: JQuery, Coldfusion, RIA

One of the more repetitive tasks a server side developer encounters is displaying the results from a query. This is traditionally in the format of a table that displays the rows of data, along with any other functionality, such as paging controls and sortable headers.

I was recently commissioned to look into building a generic table display "engine", and thought I'd investigate if there were any JQuery plugins that could do the bulk of the work for me. Ideally I didn't want to have to write a whole load of script to parse sorting variables, and detect if a limit was set on the returned record set for paging.

After some investigation I ended out using the table sorter JQuery plugin http://tablesorter.com/docs/ .

This plugin allows for sortable results that you can page through, and it does not keep posting the values back and forth to the server.

Start by including the references to the JQuery libraries. I've also included references to the paging plugin, and the blue theme stylesheet.

```
  <s/cript type="text/javascript" src="/path/to/jquery-latest.js"></script>
<s/cript type="text/javascript" src="/path/to/jquery.tablesorter.js"></script>
/cript type="text/javascript" src="/path/to/ /jquery.tablesorter.pager.js"></scrip
       <link rel="stylesheet" href="css/blue.css" type="text/css" />
```

Next we will create a fake query, so that we have some records to display.

```
<!--- Create a test query. --->
<cfset variables.qOptions = QueryNew( "id, name, color" ) />

<cfset QueryAddRow( variables.qOptions ) />
<cfset variables.qOptions[ "id" ][ variables.qOptions.RecordCount ] = "1" />
<cfset variables.qOptions[ "name" ][ variables.qOptions.RecordCount ] = "Value 1" />
<cfset variables.qOptions[ "color" ][ variables.qOptions.RecordCount ] = "Red" />

<cfset QueryAddRow( variables.qOptions ) />
<cfset variables.qOptions[ "id" ][ variables.qOptions.RecordCount ] = "2" />
<cfset variables.qOptions[ "name" ][ variables.qOptions.RecordCount ] = "Value 2" />
<cfset variables.qOptions[ "color" ][ variables.qOptions.RecordCount ] = "Green" />

<cfset QueryAddRow( variables.qOptions ) />
<cfset variables.qOptions[ "id" ][ variables.qOptions.RecordCount ] = "3" />
<cfset variables.qOptions[ "name" ][ variables.qOptions.RecordCount ] = "Value 3" />
<cfset variables.qOptions[ "color" ][ variables.qOptions.RecordCount ] = "Blue" />

<cfset QueryAddRow( variables.qOptions ) />
<cfset variables.qOptions[ "id" ][ variables.qOptions.RecordCount ] = "4" />
<cfset variables.qOptions[ "name" ][ variables.qOptions.RecordCount ] = "Value 4" />
<cfset variables.qOptions[ "color" ][ variables.qOptions.RecordCount ] = "White" />
```

We have to change the way we build the table code slightly, as the JQuery plugin is expecting certain field naming conventions. Give your table a class name of tablesorter. This is the style that the JQuery is watching for.

```
<table class="tablesorter">
    <thead>
      <tr>
        <cfoutput>
<cfloop list="#variables.qOptions.columnlist#" delimiters="," index="variabl
            <th>#variables.index#</th>
        </cfloop>
      </cfoutput>
    </tr>
  </thead>
```

In the code above I am looping over the columnlist of the query to generate headings. I've had to change the headings to 'th' tags which I never normally use.

Next we can generate the table content, making sure it is inside a 'tbody' html tag. Simply loop over the query displaying all the results within td tags.

```
                        <tbody>
                    <cfoutput query="variables.qOptions">
                                <tr>
                          <td>#variables.qOptions.id#</td>
                         <td>#variables.qOptions.name#</td>
                        <td>#variables.qOptions.color#</td>
                                </tr>
                            </cfoutput>
                        </tbody>
                    </table>
```

Lastly the pager plugin is looking for a div with a class of pager. Inside this div you place your paging controls, and the value of the page recordsets that you want to offset by.

```
                <div id="pager" class="pager">
                                <form>
            <img src="addons/pager/icons/first.png" class="first"/>
             <img src="addons/pager/icons/prev.png" class="prev"/>
                   <input type="text" class="pagedisplay"/>
             <img src="addons/pager/icons/next.png" class="next"/>
             <img src="addons/pager/icons/last.png" class="last"/>
                           <select class="pagesize">
                 <option selected="selected"  value="10">10</option>

                            <option value="20">20</option>
                            <option value="30">30</option>
                           <option  value="40">40</option>
                                  </select>
                               </form>
                            </div>
```

This builds a one page table display that can paginate and sort with a single refresh.

A full example of this is here .