ColdBox 2.6 + AJAX (powered by jQuery) = Sinfully Easy

Posted At : May 16, 2008 4:48PM | Posted By : Matt Quackenbush
Related Categories: ColdBox, ColdFusion, jQuery

I've mentioned before that I <3 ColdBox and that I <3 jQuery . Independently of one another, they are just a pleasure to work with. Today I decided to write my first-ever jQuery -powered AJAX functionality inside a ColdBox -powered application. It's ridiculously simple to do - one might even say sinful. So easy in fact, that I decided to post a quick "tutorial" on how to handle a pretty common task with ColdBox and jQuery.

The Task: Browse a User List

Pretty much any decent-size application has a user list. When the list of users gets really big, most people add some sort of functionality to the interface that allows the list to be sorted by the first letter of the last name or sometimes even the first name. It might look something like this:

Typically speaking, when you click on whichever letter you want to sort by, a brand new page will be loaded, and the sorting will be done by passing the letter to the application server in the querystring, then passed on to the database server to retrieve the appropriate records. I like to save bandwidth where I can, so I decided to update an application I'm working on and make use of a little bit of AJAX so that only the actual list is replaced/refreshed, rather than loading an entire page.

ColdBox: noLayout=true

ColdBox's setView() method has an optional argument named "noLayout". By default this argument is set to 'false'. However, for our little AJAX operation, we want to change that to 'true', because we want only our list of names to be rendered, and not the entire layout. (Remember, the layout is already on the screen, along with the default list and the alpha-select list as shown above.)

There are a couple of ways you can have ColdBox take care of this request. One way is to have separate methods; one for rendering the full page (including layout), and one for rendering only the actual list itself. Another possibility is to have a single method that handles the request, and use an "isAjax" flag to decide whether or not to render the whole layout. My personal preference is the latter, because it allows me the luxury of never messing with the HTML in the view once I've got it and the jQuery (shown below) in place. So, here is a snippet of how I handle this flag in my controller.

```
1. if ( NOT event.getValue( "isAjax" , false ) ) {
2. event.setView( "nameSelect" );
3. } else {
4. event.setView( "nameSelect" , true );
5. }
```

The first line determines if 'isAjax' is false, which is set to be the default value. If it is false, then I set the view without providing the "noLayout" argument, which as we noted above defaults to false. However, if 'isAjax' is true, I set the view and supply a value of true to the "noLayout" argument.

That is all that is required! We're now done with the ColdBox half of our update! Schweeeeeet!

jQuery: The Beauty of Unobtrusive JavaScript

For years I shied away from using JavaScript in my applications. There were two things about it that I just absolutely hated:

1. Mucking up my HTML with JavaScript code
2. The ease with which a user can disable or otherwise get around it, which would break the application (for that user)

Then I found jQuery. Now I can write a comparatively small bit of code, *and* I can leave my (x)HTML alone to do what it is supposed to do. Our little alpha-sort list is no different. I'm not going to touch the HTML at all. This way, if the browser does not support JavaScript (or it is disabled), the application still works perfectly fine. However, if JavaScript is supported, we get the bonus of using far less bandwidth and execution time. So, without further ado, here is our quick and dirty jQuery:

```
1.  $(document).ready( function () {
2.  $( "a.select-letter" ).click( function () {
3.  var theURL = $(this).attr( "href" )+ "&isAjax=1" ;
4.  var thisL = $(this).attr( "class" );
5.  var thisL2 = "" ;
6.  // start the 'loading' image
7.  $( "#ajax-load" ).removeClass( "hide" );
8.  $( "a.select-letter" ).each( function () {
9.  thisL2 = $(this).attr( "class" );
10. if ( thisL2 != thisL ) {
11. $(this).removeClass( "current-letter" );
12. } else {
13. $(this).addClass( "current-letter" );
14. }
15. });
16. $.get(theURL, {}, function (data) {
17. $( "#nameSelect" ).html(data);
18. });
19. $( "#ajax-load" ).addClass( "hide" );
20. return false ;
21. });
22. });
```

Since the purpose of this post is not to teach you the basics of jQuery, I'm merely going to highlight a couple of things that we're doing.

First, on line 3 we're grabbing the URL from the 'href' attribute of the letter that was clicked on, and we are appending our 'isAjax' flag to the querystring, setting it to true (or 1).

Next, on lines 8-15, we're removing the "current-letter" class from all of the letters that were not clicked on, and adding it to the one that the user clicked. (This is a class that highlight's the first letter of the current list. Completely optional, of course, but I like it.)

Then on lines 16-18 we're making use of jQuery's $.get() Ajax method to run our ColdBox event (request) and update the "nameSelect" div with the new list.

Lastly, on line 20, we're telling the browser "hey, don't follow the link you just clicked on", because we've already handled the request via Ajax.

Summary

It's that easy. Sinful. Delightful. I so totally <3 ColdBox + jQuery. :-)