Making The View: Part 7

Posted At : December 30, 2006 10:23 PM | Posted By : Cutter
Related Categories: Development, ColdFusion, Making the View, Model Glue:Unity

OK, so let's get interesting. We've covered creating a base template in **XHTML**, properly divided our display declarations from our content by using **Cascading Style Sheets**, and pinpointed some areas within the base template which should be dynamic. Finally, in our **previous tutorial**, we brought all of this together in a reusable template, which we demonstrated by using it as a custom tag. Now we're going to really start having some fun, by redefining our view within some frameworks. Now, I'm no expert. There are probably a few dozen ways to properly handle the view within the different frameworks, so any other suggestions are welcome as always. We're going to start off with **Model Glue:Unity**, while I catch up on the changes to **Mach II** and **Fusebox**.

You'll begin by creating a new application from the modelglueapplicationtemplate. (For thorough information on working with Model Glue you should check out the **Model Glue Documentation**, and for a brief overview you can check out **my presentation** .) From there the rest is actually pretty easy. To begin, there is already a 'template' file within the modelglueapplicationtemplate, dspTemplate. Now it comes down to some adjustments. First, let's look at how you can pass your 'dynamic' content values. You can define these items through your ModelGlue.xml file:

ModelGlue.xml

```
...
<event-handler name="page.index">
 <broadcasts />
 <results>
  <result do="view.template" />
 </results>
 <views>
  < include name="body" template="dspIndex.cfm">
   <value name="pageTitle" value="Home Page" />
   <value name="pageKeywords" value="My Site, Home Page, coldfusion, programming, XHTML, javascript, css" />
   <value name="pageDescription" value="The My Site Home Page is the gateway into My Site, where many interesting things are bound to be discovered." />
   <value name="additionalStyleSheets" value="home.css" />
   <value name="additionalScripts" value="myajax.js,calcfunct.js" />
   <value name="siteSection" value="Home" />
  </include>
 </views>
</event-handler>
...
```

You see all of the familiar variables that we had used within our previous tutorial. Now it's time to see how they get called from within our display template. At the start of our dspTemplate.cfm file we'll test for, and set, our variables:

dspTemplate.cfm

```
...
<cfset variables.pageTitle = "My Site: " />
<cfset variables.pageTitle = variables.pageTitle & viewState.getValue("pageTitle","Home") />
<cfset variables.pageKeywords = viewState.getValue("pageKeywords","") />
<cfset variables.pageDescription = viewState.getValue("pageDescription","") />
<cfset variables.additionalStyleSheets = viewState.getValue("additionalStyleSheets","") />
<cfset variables.additionalScripts = viewState.getValue("additionalScripts","") />
<cfset variables.siteSection = viewState.getValue("siteSection","") />
...
```

The getValue() method of the viewState is kind of like using *cfparam* in that it looks for a variable and serves it up, or lets you define a default if it isn't present. After this the majority of the template is unchanged, except for those areas where we begin to fill in content. Let's say you're writing your home page. Well, in Model Glue it all starts with the ModelGlue.xml, which we showed you in our first code block. From that code you'll notice the *include* tag to set your 'body'. Within your dspTemplate.cfm you'll need to verify the existence of 'body', and then display it if it is available:

dspTemplate.cfm

```
...
<cfif viewCollection.exists("body")>
```

```
 #viewCollection.getView("body")#
</cfif>
...
```

And...that's it. Now, there are other ways to do it of course. You might only have a single *value* tag to every *include* containing some page id that is passed to the controller on page load to pull your dynamic template bits from a database to then pass back to your template. Maybe you'll put the variables within your content template? There are many ways, this was just one.

That's all for now. Next time around we'll cover one of the other frameworks, which one will depend upon which one I get back up to speed on first. Questions? Comments? War stories? Let me know.