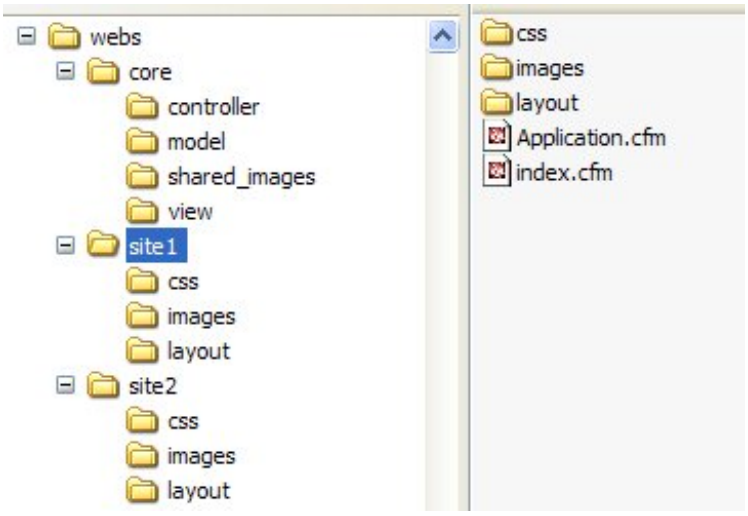
Sharing Lyla Captcha Across Applications

Posted At : September 16, 2009 12:15 PM | Posted By : Mark Kruger
Related Categories: ColdFusion

Here's an interesting problem we had to solve recently. A customer came to us with a suite of ecommerce sites on a single server. The sites were set structurally with a core set of code that supported all the sites and then individual templates that handled the layout and design. This is actually pretty common. The folder structure allowed for site specific stuff to go in the site folder while all the common stuff (everything but specific images and layout stuff) went into the site folders.



The application file specific to each site set up the variables needed for that site, then all of the heavy lifting code was called from the "core" folder using includes, custom tags or CFCs. The idea here is to be able to affect the application code of all 50 sites on the server with a single deployment. This is an idea I endorse although there are other ways of doing it. For the scope of this suite of sites it seemed an acceptable solution.

The problem came when we wanted to run code directly from *outside* the application (meaning the core) without first running it *through* the application.

The task was to use **Lyla Captcha** to protect some forms that appeared on every site. Now in case you don't know about Lyla, the code (at least the code we have) uses the application scope. It creates an image "on the fly" with an embedded text string in it and through the magic of cfcontent it is delivered to the page. Since the forms we were using were shared by all the sites we obviously wanted to put Lyla into the core. But the way the site1 folder relates to the core makes this difficult. Why? Let's look at the layout of these folders and applications. For example, application.cfm (or .cfc) in site1 might have something like this:

```
<!--- Using application.cfm --->
<cfapplication name="site1" ...../>

<!--- Using Application.cfc --->
<cfscript>
this.name = site1;
</cfscript>
```

We'll talk more about the "name" of the application a bit later. Meanwhile, here's typical code in the index.cfm file of site1.

```
<!--- include somethimg from core --->
    <cfinclude template="/core/view/blah.cfm"/>
    <!--- instantiate a core object --->
    <cfset myObj = createobject("component","core.controller.blah")/>
```

This approach results in both the included core template and the core object sharing the application scope of the calling page - in other words the application scope of the site1 folder. So, if inside of myObj, the component makes reference to the variable "Application.DSN" it will see the application.DSN variable set up in site1's application.cfm file. This enables the core code to be used as a single code base for multiple applications and it is exactly what is intended by this model. Clear as mud right?

But let's turn to Lyla for a moment. Lyla works by placing a cfm file as the source of an IMG tag as in (and I'm borrowing liberally from Ray's "quick and dirty" guide here):

```
<img src="Captcha.cfm?hash=#captcha.hash#">
```

Inside of of the file capcha.cfm the code uses the object stashed in the application scope, creates an image and then uses cfcontent to deliver it:

```
<cfcontent type="image/jpg" variable="#variables.captcha.stream#" reset="false" />
```

What's the problem? Both the form and the validation handlers (the stuff that validates the form inputs and emails it or whatever) are in the core. Since the captcha is an integral part of a piece of the *core* code I am loathe to throw the Captcha.cfm file into 50 different site directories and make it a part of every subsequent site deployment - not to mention that the handlers still have to go into the core. I would end up with a mixed system - have local site folders and half core - that is more complicated to support. I need to get Captcha.cfm into the core where it belongs.

A Possible Solution Presents Itself

But wait! I noticed that there is a "shared_images" folder inside the core. A quick glance at IIS shows that the shared images folder is mapped with a virtual directory named "sharedimg" for all site deployments. This is *also* a fairly common feature of this type of setup. The core code uses images (buttons, spacers, maybe even product images) that are common to all of the sites. Developers can then easily create a library of images that can be used by accessing a single virtual directory. In other words, from the core code they can safely use:

```
<img src="/sharedimg/something.jpg"/>
```

The image is deployed along with the core and all the sites automatically benefit without the need to copy the image over and over again. Hands up if you see my possible solution. I can place Captcha.cfm into this directory and then alter my image code to look like this:

```
<img src="/sharedimg/captcha.cfm?hash=#captcha.hash#"/>
```

Now I know that the it's not *strictly* an image, but let's not split hairs. We have a solution that potentially allows us to keep all of our captcha code in the core. And this will work splendidly except for one ticklish problem. ColdFusion will not know anything about the /sharedimg "virtual" mapping. When it runs captcha.cfm it will look for an application.cfm or .cfc file in the /shared_images folder and then in /core. When it doesn't find one (not to mention the application object it needs to do its thing) it will error out.

Of course we could just throw an application.cfm file into the /core folder, but the problem is that lyla matches its hash codes to the ones created within the application *of the calling page*. In other words, *it depends* on the form and the Captcha.cfm file being part of the same application.

So how do we manage to get something called from *outside* of the application to share the application scope with it? We use the magic of the application name.

What's in a Name

You may not know this but the "name" of the application binds all of its objects and variables together. If you want the skinny on that tidbit check out my post on **Application Variables**. For now let's just say it this way: If you name 2 different applications the same they will share the same scope - changes to an application variable in one of them will result in that same variable being changed in the other. Indeed, they are the same memory reference. So how do we use this to our advantage?

The Fix

Now this is a bit messy and not a "perfect" solution. A perfect solution would probably involve rewriting Lyla to suite our application. We took the following steps. First we put Captcha.cfm in the shared_images folder. Next we altered the URL string in the img tag in the form (in the /core/view/ folder) to:

```
<img src="/sharedimg/captcha.cfm?hash=#captcha.hash#&sitename=blah"/>
```

In this case the url.sitename variable was a string specific to each site that allowed us to figure out the actual "application name" of that site. Finally, inside of Captcha.cfm we did the following:

```
<cfset variables.apname = url.sitename & "_blah"/>
<!--- after figuring out the application name from url.sitename --->
<cfapplication name="#variables.apname#" .../>
<!--- make sure our ap scope has the captcha in it --->
<cfif NOT structkeyexists(Application,'captcha')>
    ...set up capcha in application scope...
</cfif>
```

The result was that our captcha.cfm code mapped itself automatically to the application we previously created in the "site1" or "site2" folders and, if Lyla wasn't instantiated, took care of creating the application scoped object for us as well. All of our Lyla code went into the core and we had no need to touch the site folders or site deployment code. Most importantly, the roll out of this code was accomplished in one step - by deploying the core repository - instead of the need to deploy both the core and the site code as well.

Final Thoughts

Many of you will say this is not a perfect solution because it breaks some rules (design pattern rules for example). But this problem and solution scenario illustrates an axiom of web development that is often overlooked. Web applications evolve organically over time. To be cost effective it is often necessary to continually support legacy code and make improvements without rewriting large portions of the code. This is simply the nature of web development in the wild. We are often tasked with finding innovative solution to ticklish problems that still conform to a standard of practice (or at the very least don't deviate wildly from it).