Scope of the Month: CGI

Posted At : July 30, 2009 1:40 PM | Posted By : Mark Kruger
Related Categories: ColdFusion

Starting with CF 6, most scopes became structures - objects with members - but in the pre Java days of ColdFusion there were a good many differences in how various scopes behaved. Few things were objects or structures. All those neat little structure functions that are so darn useful when dealing with a scope were not invented yet. Instead we had famously ugly, workaround code with loops using lists and evaluate( ). You might remember the good old CF 4.x days when, in order to loop through form variables, you used the "special" form variable called *fieldnames* which contained all the form field names. Remember this code?

```
<cfloop list="#form.fieldnames#" index="fName">
Form Element: #evaluate("form." & fname)#<br/>
</cfloop>
```

In fact, the special "fieldnames" form element did not disappear until ColdFusion 8 - for backward compatibility reasons I'm sure. Even back then we tried to avoid the use of the "Evaluate( )" function which was an expensive tag to be sure if you were concerned about performance. I remember that Ray was famous for going apoplectic whenever anyone suggested it as a solution. But because of how each scope was sort of special, evaluate() was occasionally unavoidable. Now days, the code above can be done using object notation and it is much cleaner.

```
<cfloop collection="#form#" item="fName">
Form Element: #form[fname]#
</cfloop>
```

More to the point, the scopes in ColdFusion all behave the same way and can be manipulated using the same basic universe of functions. Actually, there is at least one caveat - the CGI scope.

CGI - Still "Special" After All These Years

One exception to the all-scopes-behave-the-same rule is the "CGI" scope. Now CGI is a useful scope for quite a few things. You can determine the IP address of the requesting device. You can check the user agent to see if the device is a smart phone browser - useful if you want to display a link to your new "mobile" application. You can check to see what port they are using and enforce SSL for a certain page or section. These and many other things can be done by unpacking the data in the CGI scope.

CGI Genesis

One of the things to keep in mind about the CGI scope is how it comes to be. CGI contains variables that are *aggregated* from more than one source. Not everything in the CGI structure (oh yes, it's still a structure) comes from the browser. Things like the user agent, query string, cookies etc all come from the browser - or more specifically from the header of the HTTP request. But things like the Path_translated (which is the physical path of the template) and Server_software (apache, IIS etc) come from the web server. I'm not even sure *where* items like certificate information and IP addresses come from but I *suspect* they also come from the web server.

Under the hood the web server actually cobbles this list of variables together by

analyzing the HTTP request and combining it with some info it "knows" about itself and the underlying networking, and then passes it as data to the ColdFusion engine which then handles the request, produces output, and sends the output back to the web server, which delivers it back to the browser (whew!). In fact, when you think about it, the CGI scope contains in all practicality *everything ColdFusion needs to process the request*. It has the location of the file being called, the query string etc. The only thing it is missing is the "content" area of the HTTP request (where things like form elements are passed).

Get to the Point

Ok, so the source of the data is varied, it is organized by the web server and passed to ColdFusion. It's still just a structure right? Well yes, it is a structure. Code like this works fine.

```
<Cfloop collection="#cgi#" item="fCgi">
    <cfoutput>#fCgi# = #cgi[fCgi]#<br/></cfoutput>
</CFLOOP>
```

You can also do those nifty things like "structKeyList()" and "structCount()" and a host of other structury things. But there *is* one important difference that you may have run across. CGI variables do not behave in a typical fashion when it comes to checking values using isDefined() and cfif logic. Consider this code:

```
<cfoutput>
#structkeyexists(cgi,'myPrettyPony')# ... produces NO
#isdefined('cgi.myPrettyPony')# ... produces YES
</cfoutput>
```

What? That's right - the "isDefined()" call returns a yes even though there is *clearly* no variable called "myPrettyPony" in the CGI scope. Not only that, if you simply checked the value of cgi.myPrettyPony inside of a logic block without doing a call to "sctuctKeyExists()" it would *not throw an error*. This is a departure from how ever other scope works. The form scope for example will throw an error if you try to check the value of something in it that is undefined.

For example, an "Undefined" Form Variable Throws an Error.

```
<!--- There is no spoon --->
<Cfif form.spoon IS 'Silver'>
    Do not try to bend the spoon....
</CFIF>
```

Whereas an undefined CGI variable is accepted like that party crasher at your family reunion who mumbles "I'm a second cousin on ... uh Bill's side of the family". We all suspect he doesn't belong but we serve him potato salad anyway.

```
<!--- structkeyExists() would say NO,
        but this code works anyway --->
<Cfif cgi.SnideComment IS 'Watch out for the vase'>
    ...cook your noodle later...
</CFIF>
```

Why is This Important?

One reason this is important is debugging. Since CGI variables are blithely treated as if they exist, there are cases where you might have trouble finding certain kinds of errors in logic. Take spelling for example (or should I say "tkae speellign fro exmaple"). If you

are including logic blocks that use CGI variables make sure and watch out for spelling. Say you do something like the following:

```
<cfif cgi.remote_adddr IS '10.0.0.1'>
    <cfabort>
</cfif>
```

This code would never trigger. Why? Because you have added an extra "d" into the variable name. When testing you might think the code is ok because no errors are being thrown. If you are like me you *rely* on such errors to help you tease out spelling issues when you get to typing too fast. So it is important to know this special case so you can account for the difference when using CGI variables.

Happy coding and as always I the Muse welcomes all *polite* comments :)