Class Compiling Fun with ColdFusion

Posted At : April 6, 2012 10:55 AM | Posted By : Mark Kruger
Related Categories: ColdFusion

ColdFusion is Java - most people know this in the abstract sense. In sales meetings with the non-initiated I speak about ColdFusion as a *layer of Java Services* like mail, networking, jdbc, and *compiling* coupled with a language and syntax that offers faster development and better maintenance. I keep the conversation firmly rooted in Java because in reality this description is spot on. With the advent of ColdFusion 10 my case will be bolstered by TomCat as well - making it even easier to sell (and frankly it's not very hard if you know what you are doing).

Since it is Java you probably already know that ColdFusion takes your CFML code and compiles it down into Java Classes. In the days of CF 6 (back when I had more hair) you could use a command line to pre-compile CFML and even save off the .JAVA files. I'm not sure if you can still do that but it was a neat trick. Every time you run a cfm or cfc file ColdFusion checks (assuming trusted cache is *off*) to see if the file has changed and recompiles it if needed. You can see this happening with a little effort. The easiest way is to go to the /cfclasses folder for the instance you are using and delete all the class files that are there. Then run a CF page. You should see class files show up for every page and each function within the page.

Knowing (or not knowing) how things *really work* is very important to a high skill set developer. It amazes me to no end when developers profess they are "uninterested" in certain things regarding the technology they work with. I can't imagine Tony Stewart being uninterested in the bore size of his cylinders or the torque of 4rth gear or whatever. I'm sure Tiger Woods has more than a passing knowledge of how golf balls and clubs are made and customized. Indeed the more broad your knowledge and the more eclectic your skill set the more likely it is that you are an effective troubleshooter. The Muse (for example) has more than a little networking, hardware and server config experience. Often this is the difference between many hours of fruitless searching and a fast "Aha!" moment. With that in mind I'd like to share a little tidbit I picked up along the way (on StackOverflow from ColdFusion/Flex developer Sean Coyne of **n42designs.com**) having to do with compiling. It started with an error I have seen many times... "Routines cannot be declared more than once". I'm sharing this because I thought the work around was unique and I had not seen it before.

Routine Declarations

In this case "routine" is synonymous with a functions. Note, I do not mean functions attached to separate cfcs. componenA.testfunction() is a *different* declaration than componentB.testfunction() - even if the 2 functions have identical names. What I mean is that no function can be declared twice within the same scope. The error is most prevalent in legacy code where you are trying to organize or refactor your code for reuse. Usually, a function is declared in one place then declared again elsewhere within the same scope (often the *variables* scope). These look like identical class declarations to CF and so it throws the error.

Example

In our file "testudf.cfm" we have the following function declaration.

```
<cfscript>
```

```
    //test function
    function test() {
        return "I'm not trying to rob you Bilbo.";
    }
    // test our function
    writeoutput(test());
</cfscript>
```

Running this script produces 2 class files:

- cftestudf2ecfm5555555$funcTEST.class
- cftestudf2ecfm5555555.class

So the compiler produces a class for the root cfm page and another one for the function declared within the page. That seems obvious and makes sense. Note, when you experiment with this you need to delete the class files and make a change (add a space or something) to the source file so that they are recompiled. Otherwise the ColdFusion server uses the class in memory and does not recompile it. So now we have our 2 files.

Working Around Routine Declarations

Here's the problem that started me down this path. The developer in question was getting this routine declaration duplication error because his error function was declared in both a commonly included UDF library file and his error template handler. So sometimes the error handler would cause errors (and that's never a good thing). To fix it he tried adding something like the following code to his error handler.

```
<!---MAK: In one file he would have--->
<cfscript>
    //test function
    function test() {
        return "I'm not trying to rob you Bilbo.";
    }
    // test our function
    writeoutput(test());
</cfscript>
<!---MAK: Then in the other file.....--->
<cfscript>
    if(NOT structkeyExists(variables,'test')) {

        function test() {

            return "I'm not trying to Rob you Bilbo";

        }
    }
</cfscript>
```

Basically he was trying to circumvent the declaration if the function was already declared. But this didn't work. In fact he got the exact same error.

A Fix?

Actually there *is a fix* suggested by Sean. If you move the function to its own file and include it the code above will work. It looks like:

```
<!---MAK: In one file he would have--->
```

```
<cfscript>
    //test function
    function test() {
        return "I'm not trying to rob you Bilbo.";
    }
    // test our function
    writeoutput(test());
</cfscript>
<!---MAK: In second file we now have --->
<cfif NOT structKeyExists(variables,'test')>
    <cfinclude template="incTestUdf.cfm"/>
</cfif>
```

This works because in one case (I think) the compiler chokes trying to produce a compilable source code file from whole file - including the declarations (both of them), but in the patched-up case it is able to separate the compiler operations and source files. It certainly produces 2 separate files. Yet if you try to simply "include" the file without the check to see if the function exists you still get the error - as if the compiler were examining it inline.

Conclusions

It's an interesting study in how CF works under the hood, but this work-around fix is really just piggy-backing on a nuance of how CF compiles. As a coding standard I would not recommend such a hackneyed approach. The Muse is all about getting things done and I'm well aware that work-arounds are necessary at times. Still, the problem here was structural. In the end the developer moved his udf library to a single location and called it from within the application file. I think that is a much better approach.