Compiling Java With ColdFusion - Development Tip

Posted At : October 25, 2011 11:25 AM | Posted By : Mark Kruger
Related Categories: ColdFusion

One of the things that separate advanced developers from intermediate (at least around here) is the use of Java. Most advanced CF developers know that if ColdFusion doesn't provide *precisely* the functionality you are looking for you can usually find something in Java that will do the job. Now I am *not* talking about petulant PHP or Java developers who are being "forced" to write in ColdFusion. Such developers tend to write rather awful code that jumps through hoops in order to make ColdFusion do something the PHP way or the Java way. These folks never figure out how to take advantage of ColdFusion strengths and they are often left with code that must be refactored. Still, ColdFusion and Java are blood brothers. It's axiomatic that if CF can't do what you want, Java can usually come to the rescue. In this post we will discuss a method to treat your Java development just like your ColdFusion development - compiling it automatically at at application refresh for easy development. But first, let's talk about why working with Java can be a bit tricky for ColdFusion folks.

You already know it is possible to work with native Java objects directly within your CF page. It can be a little clunky to get started but it is often well worth the effort. I've made that point in many previous posts like **this one** on a directory list that performs a tad better than "Cfdirectory" and my post on using Java sample code to create requisite Java objects in your CF page - found **here**. When a Java object can't be used directly in a page (sometimes it's simply too complex or implements things you can't mirror in CF), you can use the "wrapper" approach. Create a class that acts as an interface to the stuff you want and expose methods to your CF. Typically, given a few samples, even a novice Java programmer can accomplish this.

One problem with turning to Java is that it tends to cause headaches with the flow of development for many CF developers. ColdFusion folks are used to making changes to *scripts* and then running the changes. They don't really think about what's happening under the hood. Yes, they know that ColdFusion is invisibly compiling the code into Java byte code somewhere in the depths of the file system, but in reality programming in CFML is very iterative allowing for small changes that are visible immediately at runtime without any intermediate process (it's there - simply obscured by the ColdFusion application engine).

Not so with Java. A Java developer has to compile his code to classes. He then has to load his classes or get them on the class path or whatever in order to be seen and utilized. Of course Java developers have some really cool tools that abstract this as well - and Java development is *probably* more dependent on things like debuggers which allow them to intercept variables and objects "in mid-stream". This "Java'esque" dev environment can find itself at cross purposes - or at least alien - to the ColdFusion development environment.

What most CF folks do is divide the labor between two environments. They compile jar files and then put them on the class path in the Java args (or dump them in the /lib directory) and restart CF - then begin testing their changes. This slows down the "iterative" nature of ColdFusion programming. One solution you probably already know about is the class loader - the ability to load an arbitrary class in CF (based on a .class or .jar file) using the the Java class loader (which uses coldfusion.runtime.java.javaProxy and java.net.URLClassLoader). Ben Nadel has a

great post on that approach found **here**. Using the class loader the CF developer can compile his classes and have them loaded directly at runtime. That solves one problem, but you still have to compile process and division of labor right? You are still developing one set of things "the Java way" and another "the ColdFusion way".

The Solution

It turns out that with another bit of code you can not just *load* but *compile* your .java files from within ColdFusion. In the example I saw the developers had both .Java and .cfm/cfc files open in the same Eclipse environment. They were able to make changes to either types of files and then reload the page. The changes were automatic because CF recompiled the class file before loading it. Don't believe it? Check out this bit of sample code:

```
<cfscript>
    // here's our compiler
    //NOTE: this could also be createObject("Java", "com.sun.tools.Javac.Main");
myCompiler = createObject("Java", "com.sun.tools.Javac.Main", server.rootdir &
"/lib/tools.jar");
    //We need to instantiate some Java classes.
    //We need a Java case srting writer to pass to printWriter
    myJavaString = createObject("Java", "java.io.StringWriter").init();
    //create our print writer. Compiler needs this class.
myStringClass = createObject("Java", "java.io.PrintWriter").init(myJavaString);
    // we need a class object
    myClass = createObject("Java", "java.lang.Class");
    //now we create a Java array for some args we will be setting
myArray = createObject("Java", "java.lang.reflect.Array");
    //output file needs a path as well.
    myOutputClassFile = createObject("java",
"java.io.File").init("c:\blahoutput\blah.class");
    //Now we use our array to populate a map of arguments.
    myArgs = myArray.newInstance(myClass.forName("java.lang.String"), 7);
    //set all our variables now
    myArray.set(myArgs,1,'-d');
    myArray.set(myArgs,2,'c:\blah\lib\');
    myArray.set(myArgs,3,'-classpath');
    myArray.set(myArgs,4,'c:\blah\lib\');
    myArray.set(myArgs,5,'-sourcepath');
    myArray.set(myArgs,6,'c:\blah\');
    myArray.set(myArgs,7,'blah.Java');
    //now we can compile
    result = myCompiler.compile(myArgs,myStringClass);
    //strangly - the results are actually written back to our StringWriter object.
    if(result IS NOT 0) {
        writeOutput("<pre>" & myJavaString.toString() & "</pre>");
    }
</cfscript>
```

Like most Java code ported to CF it's more complex that it needs to be (Java is so darn *wordy*), but it should be easy enough to follow. All the set statements up to the myArray.set() methods are basically various class objects neeeded to get us to the compile function. The compile() function takes an array of values. Any value beginning with a "-" is a switch for the value that follows (think command line as in -d c:\blah\lib -classpath c:\blah\lib). followed by a list of files to compile - in this case only one, blah.java. The code above was assembled by extracting it from a complex CFC that was capable of compiling all the files in a given directory. I haven't tested it but the

basics are there. If you happen to test it and debug it - please post your changes and I'll make sure and update it so it becomes more useful as a sample.

To summarize, in order to make changes to ColdFusion *or* to the Java classes being used, the developer only has to make his changes to source files (of either type) in eclipse and save the file - then reload the page (or reload the application). The underlying code recompiles the Java classes and then "reloads" the class (using the Java.net.URLClassLoader) to see his changes in effect.

Caveats

This is useful code *in development*. I do not believe such code should be ported to production. I really don't think manually firing up the compiler is a great idea on a production system, and the class loader has demonstrated some problems with releasing references causing heap problems over time (although there are rumors this is greatly mitigated in recent JVM versions). But for development purposes where Java is a significant dependency I think this is some really clever code! Tell the Muse what you think.

Also, a shout out to Twin Tech's **Jesse Dailey** for the code samples and demo - and for letting me write about it. Thanks Jesse.