Iterations, Comparisons and JavaCast (Oh My)

Posted At : November 26, 2007 3:13 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Optimization, Coldfusion 8

Testing Gurus like Dave Watts and Robi Sen will tell you that iterative tests are only interesting in a passing academic sort of way. Indeed if you are testing a real world application don't bother setting up a fancy-pants loop with 50,000 iterations to see if you should use "IS" or "EQ". If you are going to test then take the time to test real world operations in your application. But while we are on the subject of iterative tests I got to pondering what goes on under the hood. For example, what happens when you do "var1 IS var2" in a CFIF.

Because Coldfusion is typeless it has to figure out how to compare these values. It converts them into doubles or strings (or whatever) and then runs the comparison function for that object type. Here's an example that illustrates the point. If you run the following code:

```
<!--- set var 1 to --->
<cfset var1 = true/>
<!--- compare var1 --->
<cfif vari1 IS "True">
    I'm true!
</cfif>
```

Coldfusion creates a Boolean value called "var1" and sets it to true. Then it converts the Boolean value from true to the string "True" using whatever logic the engine has proscribed for that function. Then it runs the string comparison operator to see if the string "TRUE" compares (without regard to case) to the string "True". Along the way it might try a few other things in a try/catch block - like double or float.

When I'm working with CFIF's I always try to match type - under the assumption that it might help (or maybe I'm just anal that way). I would not do the following:

```
<cfset var1 = 21/>
<cfif var1 IS "21">
    I'm of age! whoopee!
</cfif>
```

When I look at that I see a string being compared to an int. Instead I would do the following:

```
<cfif var1 IS 21>
I'm of age! whoopee!
</cfif>
```

I know that this makes no difference from a performance standpoint, but someday there might be ways to specify the type comparison you are after. In a very large application this might be worth trying - so I want to be prepared :)

Meanwhile Back at the Java Ranch

Anyway, I got to thinking about the topic above and the use of Javacast. What if, using Javacast, you could influence the underlying engine as to what kind of type comparison is appropriate. For example, what about this code:

```
<cfset var1 = 32.45/>
    <cfset var1 = Javacast("double",var1)/>
    <cfset var2 = 31.73/>
    <cfset var2 = Javacast("double",var2)/>
    <Cfif var1 IS NOT var2>
        I'm not the same!
    </CFIF>
```

Would this approach be passing variables with enough information to cause the compiler to "cut to the chase" and use the correct “double” comparison operator? After all we have created two variables that are each of the Java "type" of double. The compiler should be able to compare them without any divergance to string comparison, boolean, floats or whatever. I set about trying to find a way to test this. I created 2 test scripts.

Conventional Comparison

```
<cfset stringToCompare = '3,5,3,5,6,76,3,67,34,12,6,7,43,5,78,5,34,6,54,7,45,76,8,4,6,43,5'>

<Cfset stringArr = listtoarray(stringToCompare)/>

<cfset listA = '3,2,5,3,23,5.2,3234,34.2,4,3.2,54,64,34,34'/>
<!--- set to array --->
<cfset listArr = listtoarray(listA)/>

<cfset ticks = gettickcount()/>
<!--- loop n times --->
<cfloop from="1" to="35000" index="x">
    <!--- grab one of the items --->
    <cfset item = listArr[randrange(1,arraylen(listArr))]/>
    <Cfset item2 = stringArr[randrange(1,arraylen(stringArr))]/>

    <Cfswitch expression="#item IS item2#">
        <cfcase value="NO">y,     </cfcase>
        <cfcase value="YES">n,        </cfcase>
    </CFSWITCH>
</cfloop>
<Cfset ticks = getTickcount() - ticks/>
<h4><cfoutput>NO CAST #ticks# </cfoutput>milliseconds</h4>
```

This one is pretty straightforward. I'm simply trying to compare 2 numbers chosen randomly from 2 arrays. I ran some tests with this on a lightly used CF 8 dev server and I got around 260 milliseconds. I wanted to see if using javacast would shave off a few milliseconds. So my next attempt looked like this:

```
<cfset stringToCompare = '3,5,3,5,6,76,3,67,34,12,6,7,43,5,78,5,34,6,54,7,45,76,8,4,6,43,5'>

<cfset stringArr = Javacast("float[]",listtoarray(stringToCompare))/>
<!--- string to find items from --->
<cfset listA = '3,2,5,3,23,5.2,3234,34.2,4,3.2,54,64,34,34'/>
<!--- set to array --->

<cfset listArr = Javacast("float[]",listtoarray(listA))/>
<cfset ticks = gettickcount()/>
<!--- loop 1000 times --->
<cfloop from="1" to="35000" index="x">
    <!--- grab one of the items --->
    <cfset item = listArr[randrange(1,arraylen(listArr))]/>
```

```
    <Cfset item2 = stringArr[randrange(1,arraylen(stringArr))]/>
    <!--- check to see if it blows up --->
    <Cfswitch expression="#item IS item2#">
        <cfcase value="NO">y,
            <!--- --->
        </cfcase>
        <cfcase value="YES">n,
            <!--- --->
        </cfcase>
    </CFSWITCH>
</cfloop>
<Cfset ticks = getTickcount() - ticks/>
<h4><cfoutput>Array CAST #ticks# </cfoutput>milliseconds</h4>
```

As you can see I'm casting into Java arrays of "floats". What was the result? I *gained* about 30 or 40 milliseconds to the operation. This code ran in around 300 to 310 milliseconds. The increase was not due to the cast operation since that occurs before I begin counting. What can account for the increase? I don't know - but I suspect that Coldfusion is still banging around trying to find the right comparison operator. Perhaps it used double and recast everything anyway. Switching my javacasts to double didn't help however.

I also tried the same code with strings:

```
<cfset stringToCompare = 'a,d,c,e,g,e,yd,3,f,a,d,e,a,y,d,e,a,y,33,ad,alk'>

<cfset stringArr = Javacast("string[]",listtoarray(stringToCompare))/>
<!--- string to find items from --->
<cfset listA = 'a,d,c,e,g,e,yd,3,f,a,d,e,a,y,d,e,a,y,33,ad,alk'/>
<!--- set to array --->

<cfset listArr = Javacast("string[]",listtoarray(listA))/>
<cfset ticks = gettickcount()/>
<!--- loop 1000 times --->
<cfloop from="1" to="35000" index="x">
    <!--- grab one of the items --->
    <cfset item = listArr[randrange(1,arraylen(listArr))]/>
    <Cfset item2 = stringArr[randrange(1,arraylen(stringArr))]/>
    <!--- check to see if it blows up --->
    <Cfswitch expression="#item IS item2#">
        <cfcase value="NO">y,
            <!--- --->
        </cfcase>
        <cfcase value="YES">n,
            <!--- --->
        </cfcase>
    </CFSWITCH>
</cfloop>
<Cfset ticks = getTickcount() - ticks/>
<h4><cfoutput>Array CAST #ticks# </cfoutput>milliseconds</h4>
```

Interestingly, when casting my arrays to Java “strings” the code actually performed a bit better - about 60 to 80 milliseconds over the code without the cast in it. On average the code ran in about 1400 to 1600 milliseconds, so this is no great improvement.

What can I learn from this? For one thing, fiddling with Javacast doesn't help when trying to optimize comparison code. This may be because CF ignores the casting but is

impacted by the additional properties that are passed with a Java object, or it may be because the specific operators for the object are slower than the primitive type comparisons that Coldfusion does for you after reducing the variables to the lowest common denominator.

The second thing I learned was that my efforts in optimization are probably better spent elsewhere. CF already does an admirable job in this area.