Making String Comparisons Faster Using Javacast

Posted At : November 30, 2007 4:28 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Optimization

In my **Previous Post** on the subject of variable comparison I tried to use JavaCast() to influece the comparison operator being used. I was attempting to keep Coldfusion from having to ask "Is it a float" or "Is it a string" or "why doesn't Ray Camden ever win People's 'sexiest man alive'". I thought that if I could pre-cast variables into Java objects, Coldfusion could use that information to cut to the chase and *know* exactly which operator to use without testing. Alas, that proved to be false hope. If you read the post you will already know that the difference between comparing typless and typed data in Coldfusion is not worth the effort.

Well as usual, smarter people than I (than me?) read this blog. The amazing **Mark Mazelin** (who's payment gateway API looks very interesting) suggested that I try to use the comparison operator directly, as in "object.compareTo()". I sometimes forget about all the nifty methods and properties attached to these objects when they are created. I set about testing his idea.

First I tried my float code.

```
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.compareTo(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>
```

I didn't expect much and I got less than I expected. This actually increased the count from 300-400 milliseconds up to 500 to 600 milliseconds. Obviously Coldfusion's internal comparison is better than the comparison operator exposed by "compareTo()" - at least for floats.

String Bonanza

Finally, I tried compareTo() on the variables cast as arrays of Java String objects.

Here's my code:

```
<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.compareTo(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>
```

To my great surprise this caused a significant decrease in the time factor. In fact the time went from around 1400-1600 milliseconds to *less than 500 milliseconds*. I shaved a full second off the time to process. I was scratching my head to explain this until I thought of case sensitivity. "Aha" I thought. The Java operator is using a *case sensitive* comparison which is naturally faster because it doesn't have to compare the string against both upper and lower case values. Secure once again in my brilliance I changed the strings to all alpha characters. I made sure that the first array of strings was upper case and the second array was lower case. Theoretically this would cause all the comparisons to evaluate as false.

The Lesson

To my surprise, they all came back as *true*. The "compareTo()" method in this case is actually the *case insensitive* operator. It gave me exactly the same result as my "variable1 IS variable2" syntax. The lesson here is that compareTo() for strings is actually faster than the "IS" or "EQ" operator. How do you use that knowledge? Well, I would say there are very few cases that warrant going to this length. If you have a large list or array of string values that you are comparing against a single value, I suppose it would not take any *extra* effort to implement this Java method. As with so many optimization techniques that come from the theoretical world or the test lab, it's not terribly practical. If you *really* want to optimize your code pay attention to the database and planning and don't get too bogged down in trimming 50 or 100 milliseconds out of your application here or there. That's my take.