Java Based Directory List

Posted At : October 26, 2006 6:15 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Tips and Techniques, Coldfusion Troubleshooting

This is a follow post to my previous post on **cfdirectory as a bottleneck**. A helpful muse reader of that post named Daniel Gracia sent me some Java code that builds a directory list. The code itself is a call to the core io.File class. It takes a directory path and returns an array of files and folders mixed (with the folders identified with a period). His claim was that it performed faster than Cfdirectory. His claim is 100% true, but there are some nuances to it. I ran a few tests and here is what I found.

The Code

Here is my test script:

```
<Cfsetting requesttimeout="200">
<cfscript>

currentDirectory = expandpath("../folder_With_15000_Images_In_It/");

function dirList(path) {
    return createObject("java","java.io.File").init(Trim(path)).list();
}

ftime = gettickcount();
    mylist = dirList(currentDirectory);
ftime = gettickcount() - ftime;

nfTime = gettickcount();
    nfList = createObject("java","java.io.File").init(Trim(currentDirectory)).list();
nfTime = getTickcount() - nfTime;
</cfscript>
<!--- Use CFDirectory as a baseline for comparison --->
<cfset tm = getTickcount()/>
    <cfdirectory action="list" directory="#currentDirectory#" name="test">
<cfset dirTime = getTickcount() - tm/>
<cfoutput>
<h4>#ftime# milliseconds for java using a function</h4>
<h4>#nftime# for Java outside of function</h4>
<h4>#dirTime# for CF</h4></cfoutput>
```

The results where no contest. The Java code performed in under 200 milliseconds consistently while the Cfdirectory call took 4 seconds - a forty-fold increase. Those numbers come from a test using local storage (meaning a physically attached disk - not a UNC path). Wrapping the call in a function is about 50% more expensive than calling it directly. In other words if the direct call is 100 milliseconds the function call will be around 150 milliseconds.

UNC Paths

To test it further I moved it to a different server and used a UNC path.

```
<cfscript>
    currentDirectory = "\\server\share\folder_With_15000_Images_In_It\";
</cfscript>
```

As you might expect running this against a UNC path takes appreciably longer. After all

it has to use the network redirector to locate the resource and make the list call. I was surprised to find that the results using a UNC path were *dramatically in favor or using the Java method*. In my tests the function wrapped Java method to the same list of 15000 images using a UNC path took around 3.5 seconds on average. The direct call was around a half a second (500ms) - a dramatic reduction.

That got me thinking. Why did that second call take so much less time (fully 1/7th of the time of the function call). Then it hit me - I was calling the same function for the same directory list two times in a row - once from with the function and immediately afterward outside the function. The function wasn't having such a tough time - it was just the first in line. Once the directory list had been built by the io.File class it was likely cached or otherwise available for that second retrieval operation. I reversed the order and called the direct method first. The result? They were almost dead even at 500 milliseconds. This indicates to me that the function wrapper has (at best) a negligible impact.

As for using Cfdirectory through a UNC path - my only advice is *don't do it!* Cfdirectory took 59 to 63 seconds on average to return the data. In case you are guessing that CF takes most of that time preparing the Coldfusion query object that is returned by cfdirectory, let me remind you that a cfdirectory call for the same directory of 15000 files took only 4 seconds when calling it as local storage. The problem is somewhere else - how the data is chunked or verified or something.

Conclusions

My conclusion is, if you are dealing with large file structures - especially across the network, and you don't need the *other* things that Cfdirectory provides (like dateLastModified), use the pure java solution and call it directly. What could go wrong?