<CFLOCK> and Synchronizing Data

Posted At : May 9, 2005 12:27 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Optimization, Coldfusion Tips and Techniques, Using Coldfusion Tags

Locking among CF developers arouses the same sort of Ire as the Linux Vs. MS debate. There are CF developers who believe that locking is unnecessary in *most* cases and locking everything is an unnecessary waste of time and effort. There are others who believe that everything should be locked without regard to any examination of the data or its significance. Both camps have salient points and I do not fault either of them for their viewpoint. In our case we err on the side of caution. We lock every write for the session and application scope and we lock "most" reads of the session scope. That's our standard and I only mention it to get it out of the way. I do not intend write an apology on wether you *should* lock or not. Instead, I think we should examine how locking syntax varies and how it used appropriately (or inappropriately).

In the most recent spate of discussion on the **CF-Talk** list a group of guru level developers - **Matt Roberston**, **Adam Churvis** and James Holmes - went round and round about locking. James Holmes produced the sample code I'm going to use to demonstrate the usage of the CFLOCK tag - so thanks Jim. Create 2 templates and put them in a folder with an application.cfm page (one that uses the <cfapplication> tag). Here are the 2 templates.

Template 1

```
<h4>Testing the application Scope</h4>
<cflock timeout="10" throwontimeout="yes" type="exclusive"
scope="application">
<cfset APPLICATION.MyStruct = StructNew()>
<cfset APPLICATION.MyStruct.MyText = "One">
<cfset APPLICATION.MyStruct.MyNumber = 1>
</cflock>
<!--- second block --->
<cflock timeout="10" throwontimeout="yes" type="exclusive" scope="APPLICATION">
<cfoutput>
Start Sleep @ #timeFormat(now(), "HH:mm:ss")#<br />
<cfflush>
<cfset APPLICATION.MyStruct.MyText = "Two">
<cfset createObject("java",
"java.lang.Thread").sleep(10000)>
Wake up @ #timeFormat(now(), "HH:mm:ss")#<br />
<cfset APPLICATION.MyStruct.MyNumber = 2>
</cfoutput>
</cflock>
```

Template 2

```
<h4>Application</h4>
<p>
<cfoutput>Text is #APPLICATION.MyStruct.MyText#; Number is
#APPLICATION.MyStruct.MyNumber#</cfoutput></p>
<p>
```

You will notice the first template has a "sleep" function in it that lets us delay the completion of the lock. This will allow us to *force* locks to compete against each

other. You will notice that for the sake of this exercise we are only using the application scope. Here's how it works. Open the first template. It's job is to set application variables, but the second "locked" set command has a "sleep" function that delays it's completion. Open the second template and it will display the variables set by the first. If you open the second template within a few seconds of the first you will notice that the "mynumber" variable is set to 1. Refresh it after the first template is finished and you will see that the "mynumber" variable is set to 2.

This demonstrates the "synchronisity" (or lack thereof) of the data. If it's important for an item to have only 1 value and for it to be the current value, then you have to ensure that read and write locks are in place to keep synchronisity intact. Otherwise there is the potential of triggering events erroneously. Now, if you add a read only lock to the second variable as in:

```
<Cflock timeout="10" type="READONLY" scope="application">
<cfoutput>Text is #APPLICATION.MyStruct.MyText#; Number is
#APPLICATION.MyStruct.MyNumber#</cfoutput></p>
</CFLOCK>
```

and try your test again. you will see that the second templates waits for the first one to finish and then displays a 2 - the correct number. You have succeeded in keeping your data synchronized (congrats - mom will be proud!).

One of the pitfalls has to do with that "scope" tag. The scope tag locks the entire scope for the duration of the operation. In the case of the *session* scope this is not a huge deal because it only locks THIS session. Try it - get 2 sessions running (2 different computers or browsers) and run page with a session scope lock and a timeout (change the "application." to "session."). You will see that the lock in your original session causes the appropriate delay in the second page, but has no affect on the other session. That's kind of news to me. I had thought that the "session" scope was locked for "all" sessions, not just "this" session.

As for the application scope, it is a broader scope - and it is indeed shared by all. So locking the application scope has the effect of tying up the application for the duration of the lock. Typically however, application variables are *write-once* and *Read Many* variables, where locking the scope once for the life of the application is an acceptable delay. But note, if a variable is changing and the writes *and reads* are not **both** locked, the data will not be in synch. Again, this is news to me in a way. Creating an "exclusive" lock on an application variable *only prevents other similarly locked variables from reading it*. It does not prevent an unlocked read of the var. So, if synchronisity is important, make sure and lock the reads as well as the writes. And lock them using the same scope!

What do you mean "similarly locked"

Ah... here's an interesting fact that is almost never brought up. You must use the same kind of scope when locking variables. For example, if you have used named locks in one place and scope locks in another you will not get the result you are after. Try it. Leave template 1 locked at the application level (scope="application") and replace the lock on the second template with a named lock.

```
<Cflock name="foo" timeout="10" type="READONLY">
<cfoutput>Text is #APPLICATION.MyStruct.MyText#; Number is
#APPLICATION.MyStruct.MyNumber#</cfoutput></p>
</CFLOCK>
```

You will see that the lock has no effect. The purpose of a "named" lock is to allow a developer to granularly synchronize data across locks. So, If I'm locking it here with a name of foo and reading it there with a lock name of "foo", the second "foo.read" event can't happen if the first "foo.write" event is not complete (assuming exclusive locks). I've seen code that generates a Random lock name for each lock. Such code literally has no effect on the synchronization of the data.

Named Locks and files

One place where named locks come in handy is in file writes and reads. You can use a named lock to ensure that the file is synchronized and that it is not corrupted by multiple writes - nifty. If you use the file name as the lock name you will always ensure compatible locking.

Thanks to Adam, James and Matt for their animated discussion on the topic.