Troubleshooting Scheduled Restarts

Scheduled Restart Fails

Chuck Hill

Let's take a look at what happens in a little more depth. When the appointed time comes, wotaskd sends the application a message to start refusing new sessions. The application will start redirecting all subsequent requests without a session to an instance that is not refusing new sessions. Of course, in your case eventually all are refusing and the woadaptor throws up its hands in dismay and declares "No Instance Available".

But the part of interest to us happens before that. After being told to refuse new sessions, the app quietly works away serving the users who already had sessions, waiting for them to terminate. As each terminates it checks to see if the number of remaining sessions is below the threshold (1 by default). If so it calls terminate() and shuts down the run loop. At this point, if all is well, there are only daemon threads running and the JVM stops them and exits. wotaskd notices that the app is not running and restarts it. That is if all is well.

Now, what can go wrong? The two most common problems that I have seen are (1) immortal sessions - the session count never drops to zero and (2) stuck threads that are not daemon threads. The first situation prevents the app from calling terminate() and beginning the exit process. The second situation prevents the JVM from terminating after the main app threads stop.

Immortal sessions are caused by an exception being raised from your code that prevents the session from being checked into the session store. The number one cause of this is exceptions being raised in the sleep() method of session. If you use this method, wrap it in a try...catch for throwable. Do not let this method throw under any circumstances. Immortal sessions can also be caused by exceptions being thrown from awake() and terminate() and I have no doubt there are other ways as well.

Stuck threads usually result from some sort of thread deadlock and can often result from immortal sessions when a request is made for a session that was never checked in from another thread. A similar situation can occur if your are creating threads in your application that are not marked as daemon threads and not shut down when the application terminates.

OK, all that is well and good but what do you do about this? The first thing you want is to wait for this to happen and get a thread dump of the application. There are some instructions here:
http://lists.apple.com/archives/webobjects-dev/2003/Sep/msg00362.html
Note that the file you need to edit is in /System/Library/WebObjects/JavaApplications/wotaskd.woa/Contents/Resources. Make sure that the appserver user has write permissions on the directory/file you point it to.

The next thing you want to do is to add some better logging to your application. At a minimum, I would suggest:

- Log ID when session is created
- Log ID when session's terminate() is called.
- Log the application's remaining activeSessionsCount() when session's terminate() is called
- Log start and stop of RR look in dispatchRequest() in application
- Log when session are refused and activeSessionsCount() in refuseNewSessions(boolean)
- Log when application's terminate() is called

Sending Notifications on Failed Restart

Chuck Hill

My first choice would be to fix your code. Second, it you have a long session time out and users are still working, then it could take quite a while for the app to restart (as long as the users keep working plus the session timeout). Third, if the JVM is waiting on a stuck thread there is not much you can do as any daemon thread to send the text page would have been killed already (hmmm, I think so at least). OK, so ignoring all that, start a timer as a daemon thread in refuseNewSessions(boolean) when true is passed. When the timer fires, send the text message. Or, if you like to sleep as much as I do, just call System. runtime().exit(1). If refuseNewSessions(boolean) is later called with false, kill the timer. That is just in case someone is clicking around in JavaMonitor.