

Re: JVM/Java memory footprint

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2007-01/msg02795.html>

- *From:* "Daniel Pitts" <googlegroupie@xxxxxxxxxxxxxxx>
 - *Date:* 29 Jan 2007 22:27:15 -0800
-

On Jan 29, 9:40 pm, alex...@xxxxxxxxxxx wrote:

On Jan 30, 2:56 am, "Daniel Pitts" <googlegrou...@xxxxxxxxxxxxxxx> wrote:

On Jan 29, 6:42 am, alex...@xxxxxxxxxxx wrote:

Hi,
Though I have some three four years experience in C++ ,I did not have that much oppurtunity to work in Java. Currenty I was doing some analysis for a very simple CLI and was surprised to come with a memory restriction. I found that if I use Java for developing the CLI application I will be exhausting the memory of our Application Server (AS). Just to mention the architecture ,users (telecom operators) use a metaframe server client (=something like remote desktop) to login to the AS and then open a GUI to work on it. With Java I can service only about 15 clients with the available memory . I just checked the reason for this resource crunch and found that already many Java based GUI's are served by the AS and each is taking some 25 MB or more. The first thing I thought is that adding more RAM will solve this (though this is not an easy option) . Then I understood that a 32 bit system can have just about 3 gb ram for applciations and our AS had

Re: JVM/Java memory footprint

already 4 GB with 1.5 GB VM also configured. Then I thought there might be ways to make the JVM shareable. But no. Also tried to use the flags to fine tune JVM. No go there also. Irrespective of how much you limit the actual limit is in the heap size allocated to the application and not to the private bytes of JVM.

Data from Jconsole for the Java application

Memory
Current Heap Size 4.3 MB
Max heap size 12. 2 MB (set by flag)
Committed memory 5 MB
Operating System
Committed VM – 26.7 MB

Data from Perfmon (for java.exe)

Private Bytes – 27.7 MB

I am part of a large team that develops and maintains a telecom network management software system. Also this software is used all around the globe almost in around 190 countries usually by telecom service providers. So this is a very real problem that I am speaking about. I am getting to love the simplicity of Java and of the great IDE Eclipse as much as I love the power of STL; but I am coming face to face with the main constraint of Java, its memory footprint and I guess thousands of others like me must have faced similar

Re: JVM/Java memory footprint

problems.

If so the next question is what is SUN doing about this. I could not find it in their top 25 RFE's or top 25 bugs. (http://bugs.sun.com/bugdatabase/top25_bugs.dohttp://bugs.sun.com/bugdatabase/top25_rfes.do)

1) Is this really not a problem in the outside world then ?

2) Or is it that with the introduction of 64 bit HW and serveres this will be of no significance ?

Anyway I doubt if companies like ours will adopt 64 bit HW and servers because of the costs involved in it (note this is just a logical guess, I have no experience in such decisions)

Anyway I am now forced to use C++ for the client and use Java on the server side. Fortunately most of the Java in the server side is done up as EJB's all conatined in the JBoss server thus consuming only one JVM there)

I would like to get some comments on questions 1 and 2.

Thanks
Alex.C.P

Hmm, I'm not sure I understand your setup, but it appearch you have a single server machine which runs both the Java server process and the multiple client processes?

Re: JVM/Java memory footprint

Usually, Java clients are run on separate client machines, so the number of clients doesn't effect the footprint.

Anyway, to answer your questions, I've never found this to be a problem in any of our environments, so perhaps there is some problem with your architecture/design/configuration. I think that this problem doesn't have anything to do with the 32 vs 64bit platform, but perhaps a misunderstanding somewhere.

Java does not have a tiny footprint, but last I checked, $2048\text{M}(2\text{gig})/32\text{Mb} = 64$. That means that you'd be able to run around 64 separate JVM processes on one machine. This isn't a typical situation, and it still doesn't seem to fit your conclusions.

Maybe it would help my understanding if I know what a "CLI" was. I only know it as Clear Interrupt Flag :-)

Hope this helps,
Daniel.

Hi, Let me clarify – CLI just means the simple command line interface :) .Well
the architecture is like this. We have a Application Server (AS) , which is also a metaframe server. Many operators with 'dumb' terminals can establish a session with the AS using a metaframe client. (say like using a glorified version of telnet, or remote desktop or RDP). Each time a new operator connects and invokes a java GUI or CLI a new instance of Java.exe is started on the AS. Now there is another server (say BS) which servers the AS. The business logic of the java clients in the AS is served by the EJB components in the BS. Since all these are developed as EJB's they run in JBoss container which needs only single JVM (Java.exe). Also already there are 17 JVM's already running in the AS consuming about 710 MB on average .Hope this clarifies

Regards
Alex.C.P

Well, that does help explain what you mean :-)
That isn't a typical use of Java technology. Usually you have only a handful of processes on one machine, either server processes or client processes. Rarely do you have both.

Re: JVM/Java memory footprint

Re: JVM/Java memory footprint

Perhaps your glorified Telnet would be better implemented as a connection to the Java server, rather than a connect to the machine the server runs on. It really depends on how 'dumb' your terminals are. If they are capable of running a JVM themselves, then I would go that approach.

Also note that there are different types of JVM's out there. At the very least, there is a Client JVM and a Server JVM that come standard. With Java 1.6, Sun is introducing a method that allows all of the JVMs to utilize shared memory for a portion of the footprint (standard class files, and probably some other static JVM information)

Anyway, my suggestion is to try to move the client JVM process from application server to the client machine, or at try to find a way to have each client JVM process service more than one user.

.