Runtime issues

{scrollbar}

Most of the issues you may find when running Geronimo will be at start up time; and most likely driven by some conflicting resources from the environment where Geronimo is set up.

JVM arguments

Apache Geronimo v3.0 is Java EE 6 compliant application server. With that said, it will likely run on different JVM versions however the results may be unpredictable. Geronimo v3.0 requires Java 1.6.

Another common problem related to Java is that, sometimes, certain environment variables are not defined at Java installation time. For instance, Geronimo requires **JAVA_HOME** and **JRE_HOME** to be defined before running the server. As a convenience, make sure you also add `<JAVA_HOME>/bin` directory to the system **PATH**.

You can choose one of following options to pass additional JVM arguments when starting Geronimo using `geronimo` or `startup` scripts:

1. set JAVA_OPTS environment variable: solid jeffchi@Local:~/geronimo-tomcat7-javaee6-3.0-SNAPSHOT$ export JAVA_OPTS="-Xmx256m -XX:MaxPermSize=128m -XX:+HeapDumpOnOutOfMemoryError" jeffchi@Local:~/geronimo-tomcat7-javaee6-3.0-SNAPSHOT$ ./bin/geronimo run
2. and/or append the following code to `<GERONIMO_HOME>/bin/setjavaenv.sh(bat)` file: solid if [-z "$JAVA_OPTS"]; then JAVA_OPTS="-Xmx256m -XX:MaxPermSize=128m -XX:+HeapDumpOnOutOfMemoryError" fi

Server hangs due to low entropy on Linux

When starting a Geronimo server on a Linux system, the server start seems to hang. However, if you're patient, the server will eventually start. If you're inquisitive, and dump the stack traces of the java process, you'll see something like:

"main" prio=10 tid=0x0000000040c0d800 nid=0xa79 runnable [0x00007f57a04fb000] java.lang.Thread.State: RUNNABLE at java.io.FileInputStream.readBytes(Native Method) at java.io.FileInputStream.read(FileInputStream.java:220) at sun.security.provider.NativePRNG$RandomIO.readFully(NativePRNG.java:185) at sun.security.provider.NativePRNG$RandomIO.implGenerateSeed(NativePRNG.java:202) - locked <0x00000000daad63e0> (a java.lang.Object) at sun.security.provider.NativePRNG$RandomIO.access$300(NativePRNG.java:108) at sun.security.provider.NativePRNG.engineGenerateSeed(NativePRNG.java:102) at java.security.SecureRandom.generateSeed(SecureRandom.java:495) at com.sun.net.ssl.internal.pkcs12.PKCS12KeyStore.getSalt(PKCS12KeyStore.java:477) at com.sun.net.ssl.internal.pkcs12.PKCS12KeyStore.calculateMac(PKCS12KeyStore.java:834) at com.sun.net.ssl.internal.pkcs12.PKCS12KeyStore.engineStore(PKCS12KeyStore.java:788) - locked <0x00000000d3b5a768> (a com.sun.net.ssl.internal.pkcs12.PKCS12KeyStore) at java.security.KeyStore.store(KeyStore.java:1117) ...

This problem isn't Geronimo specific.

The Sun/Oracle-based JVM is attempting to generate a pseudo-random number to be used as a seed for an SSL server socket. To generate the pseudo-random number, the JVM is reading from the /dev/random device to obtain some random information for the seed. The problem is that reads from the /dev/random device will block if the system does not have a good source of random events. So, the Geronimo server startup is blocked waiting for enough random information to be returned from /dev/random. This article may be help understand the basic issue – http://en.wikipedia.org/wiki//dev/random#Linux

To avoid the problem, you can choose to use the /dev/urandom device, instead of /dev/random, by specifying the following java property `-Djava.security.egd=file:/dev/./urandom`. For example, start the server using the following command:

solid $ export GERONIMO_OPTS="-Djava.security.egd=file:/dev/./urandom" $ ./geronimo run --long

Port conflicts

The second most common startup issue is associated to port conflicts, check no other application is using or blocking Geronimo's default ports:

Service	Port
SMTPPort	25
COSNamingPort	1050
NamingPort	1099
ORBSSLPort	2001
OpenEJBPort	4201
ORBPort	6882
AJPPort	8009
HTTPPort	8080
HTTPSPort	8443
JMXPort	9999
ActiveMQStompPort	61613

ActiveMQPort	61616

If you identify port conflicts you can use the `<GERONIMO_HOME>/var/config/config-substitutions.properties` to change any of these ports. From this configuration file you can also set a port offset and have all these increased by that amount.

Keep also in mind that personal firewalls, anti virus and spyware protection products may block some of these ports as well, even if you turn off such software sometimes those "rules" are still in effect.

Refer to the Initial configuration section for additional details on prerequisites and different configurations.

Spring version conflicts

If your application contains its own version of Spring you might see some problems deploying or running the application on the Jetty assembly. The Jetty assembly is by default configured with Apache CXF as the JAX-WS provider. Apache CXF uses Spring to configure itself. Sometimes, the Spring version used by CXF conflicts with the Spring version supplied with your application. To prevent these conflicts add the following <hidden-classes> entry to the Geronimo deployment descriptor:

xml <hidden-classes> <filter>org.springframework</filter> <filter>META-INF/spring</filter> </hidden-classes>

java.lang.UnsatisfiedLinkError: lic (Library is already loaded in another ClassLoader)

This can be caused by the same jar (such as an Oracle Driver using OCI) being loaded in two separate class loaders, each trying to load a system library.

To avoid the problem, you may try Adding JARs to the Geronimo repository and define the dependency in your deployment plan.
See Also : https://issues.apache.org/jira/browse/GERONIMO-4629

Enable multicasting for IPv4 network

To use multicasting properly in a clustered enviornment, you will need to set the system property `java.net.preferIPv4Stack` to `true` when starting Geronimo.

java.io.IOException of remote EJB on Windows

javax.naming.NamingException: Cannot lookup'/MyBeanRemote'. [Root exception is java.rmi.RemoteException: Cannot connect to server 'ejbd://localhost:4201''; nested exception is: java.io.IOException: Cannot connect to server: 'ejbd://localhost:4201'. Exception: java.net.BindException : Address already in use: connect] at org.apache.openejb.client.JNDIContext.lookup(JNDIContext.java:214) at javax.naming.InitialContext.lookup(Unknown Source) at MyClientScalabilityTest$MyClientTest.call(MyClientScalabilityTest.java:86) at MyClientScalabilityTest$MyClientTest.call(MyClientScalabilityTest.java:1) at java.util.concurrent.FutureTask$Sync.innerRun(Unknown Source) at java.util.concurrent.FutureTask.run(Unknown Source) at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(Unknown Source) at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source) at java.lang.Thread.run(Unknown Source) Caused by: java.rmi.RemoteException: Cannot connect to server 'ejbd://localhost:4201''; nested exception is: java.io.IOException: Cannot connect to server: 'ejbd://localhost:4201'. Exception: java.net.BindException : Address already in use: connect

One possibility is that the available user port numbers are being exhausted. On Windows, when a socket is closed, it goes into a TIME_WAIT state and isn't actually closed until some delay time. By default, the max user port address is 5000 and the TIME_WAIT delay is 4 minutes. So, it's not too difficult to exhaust all possible user port addresses.

You have to update the Windows Registry to change these values. Here's a Windows 2000 doc on the registry settings – http://technet.microsoft.com/en-us/library/bb726981.aspx

- MaxUserPorts controls the upper range for user ports.
- TcpTimedWaitDelay controls the TIME_WAIT delay.

Increasing MaxUserPorts (e.g. 65534) and decreasing TcpTimedWaitDelay (e.g. 30) may fix the problem.