

In this video we are looking at the profiling of an actual transaction. We have already seen the client installation and server installation in previous videos, as well as a quick verification test. Jinsight has several modes of operation when profiling. The most useful mode is the live mode, in which it connects from the client to the server over TCPIP, allowing us to selectively trace transactions or parts of applications. To do this, essentially we need to add Jinsight to the command line of the server in question. What we have here is Websphere for z/OS, version 6.0.2 and we need to go to the admin console to add our Jinsight options. Here we select *Servers* and *Application Servers*, then the server name which we wish to profile. We scroll down until we find *Java and Process Management*; we expand this tree, and go to *Process Definition*. You may find on other versions of Websphere the menu system changes or is slightly different.

On z/OS we have a choice of three JVMs we could profile. Normally, one would be profiling the *Servant JVM*, which is running the application code, but occasionally you may wish to, under special circumstances, profile the *Control* or *Adjunct* regions. The documentation covers this in more detail. We select *Servant* and then *Java Virtual Machine*. Here, I have already made the two changes required to enable Jinsight. This is a reference to the *jinsightServer.jar* file in the boot classpath and most importantly, we are adding the `-Xrun` parameter, to the generic argument JVM arguments or to the command line of the JVM. You may find that there are other parameters already defined in your generic JVM arguments. Jinsight can tolerate virtually all of them, apart from other `-Xrun` parameters. These will have to be removed for Jinsight to be able to work, as only one shared library can attach to the JVMPI interface at any one time. Here we have our option `-Xrunjinsight-zOS-31bit`, but we have added an additional option for Jinsight, which is defining a port number. One could leave this out and allow Jinsight to randomly select a server port, but then we would have to look at native *stderr.log* in the Websphere log directory to establish which port has been selected by Jinsight. So often it is easier to define a particular port and also, if we are behind a firewall, we can have that port opened. Therefore, we can be guaranteed to be able to get through the firewall and connect to the server. Here, I have chosen port 12345, so the full syntax is `-Xrunjinsight-zOS-31bit:port=12345`. The other setting we need to make, if we go back to the parent screen here, is in the environment variables. If you remember from the quick verification test we made, when installing the server component, the Websphere process on the JVM will need to easily locate the shared library. To do this, we need to set the *LIBPATH* variable. Here, I have created a new environment variable, with the name of *LIBPATH*, and the value of `/opt/zJinsight`. Websphere intelligently appends this to any existing *LIBPATH* environment variable it may find. These are the settings and I need to go off and restart Websphere for these to become active. You may find, if a mistake has been made, or the shared library cannot be located, or there may be a permission issue, that the servant region will not start. You will find in the

logs of the control region the potential reason for that. So I have already restarted Websphere, so I can close this admin console down and let's bring up the server that we are going to profile. In this case, it is the *SuperSnoop* servlet, which is one of the default applications shipped with Websphere. As you can see, it runs. What I am going to do is to close this down and we are going to open up the Jinsight client. I am going to the C: drive, find the *zJinsight* directory. Underneath, we have *jinsightlive* and we execute the batch file, which starts the Jinsight client itself. We can then hit the *File* menu, *Connect to a Java VM*, and type in either the IP address or the hostname of the server we wish to connect to and the port that we chose, or Jinsight has chosen for itself. You may find that the best way to profile is to save to a particular trace file, and we can give it any particular name that we wish and *only save to file*. If we do not select this option, Jinsight will start loading the events into memory, as the client is also the visualizer as well as the profile collection tool. If we are going to be profiling an unknown number of events, we may find that we exceed the client memory and this could seriously affect the performance of the Jinsight client or terminate it all together. So, it is best to do a two part profiling exercise, in which one traces and only saves to file, and then opens that after one has finished. You can open a new client and load the trace file selectively, therefore not exceeding client memory. We will click OK here and you see that the client window is telling us that we have a live connection to the host *zos* on port 12345. If we now bring up the *SuperSnoop* servlet, this means that we have warmed the JVM; we have already ran the transaction once. This is important as we do not wish to see in the profile class loading and application initialization. I then hit *Start One Burst* on the client and execute the transaction again. You may not immediately see events being sent down to the client as the server has an amount of buffer. If we hit *Stop This Burst*, we will then see the events coming down from the server to the client. We have 57490 events. I can now, if I wish, close this client. It will disconnect cleanly from the server, and I can then go on to profile other transactions, or, alternatively, I could remove Jinsight, restart the server, at its normal mode of operation.