Java Bytecode Instrumentation Limitations

A real transaction management product needs to follow through the transaction between different types of application-related components such as proxies, Web servers, app servers (Java and non-Java), message brokers, queues, databases and so forth. In order to do that, you need visibility to different types of transaction-related data, some of which only exists at the actual payload of each request. Java, since it is an interpreter, hides parts of the actual code implementation from the Java layer. The Java Virtual Machine (JVM) itself is written in C, therefore there are operating system-specific pieces that are not accessible from the Java later, and thus not accessible to the bytecode instrumentation (BCI). Also, different Java packages utilize native code for reasons of performance or code reuse. This native code (libraries loaded to the JVM) is not accessible by BCI since it is not written in Java.

For example: Let's say that we need some data that is part of the handshake between the Java application server and Oracle. Using BCI, you can only get access to data stored and managed by the Oracle JDBC driver. If some of the implementation is part of a .dll/so file the JDBC driver is using, we will not be able to access it by using BCI.

Another example: if we want to use features of TCP/IP packets for tracing a transaction between two servers, the actual structure of packets is not accessible from the Java layer, since it is done by operation system libraries that are utilized by the JVM itself.

Read my previous blog posts that introduce bytecode instrumentation and discuss how bytecode instrumentation affects transaction tracing.

Overall, Java bytecode instrumentation is a useful concept that makes the lives of Java developers much better by giving potential visibility into every single class and method call. The deeper the visibility, the higher the overhead. However, a transaction management solution for a production environment cannot allow itself to impact such magnitude of overhead, requiring the trace to be limited by design. A limited trace has to be tailored to every Java application, which makes implementation in real-life scenarios a much more costly task. More so, there are pieces of data that can be crucial if you want to trace transactions across more than just one Java hop, and they are available only at the lower layer than the Java code, thus not accessible by BCI and limiting the ability to trace transactions in the real world.

Posted on January 3, 2011

bci, bytecode, bytecode instrumentation, java bytecode, Lanir Shacham, transaction management, transaction tracing

← End-to-End Transaction Monitoring

First International Bank of Israel →

Transaction Management

Transaction Tracing

Archive

May 2018

April 2018

March 2018

February 2018

January 2018

December 2017

November 2017

October 2017

September 2017

July 2017

April 2017

March 2017

January 2017

December 2016

Learn

- About Correlsense
- Meet the Team
- News
- Testimonials
- Our Partners
- Customers
- Board of Directors

Engage

- Product – SharePath
- Product Features
- Case Studies
- Resources
- Blog
- Get SharePath Free

Follow

Search …

GO

Privacy Policy