

Java Bytecode Instrumentation: An Introduction



This post is not a usual one since I simply want to address a technical question: “what is Java bytecode instrumentation (BCI)” and also explain what can and can’t be done with BCI regarding the problem of transaction tracing. It’s just that I’ve been asked about it again and again, and there is a real confusion out there in the market. Vendors that ONLY do BCI (e.g., CA Willy, dynaTrace, AppDynamics, etc.) are claiming to be a transaction management solution, although there are limitations to what they can do in Java environments, and they have zero visibility to non-Java topologies.

BCI is a technique for adding bytecode to a Java class during “run time.” It’s not really during run time, but more during “load” time of the Java class. I’ll explain: Java, for those who are not familiar, is a fourth generation language, which means you write Java code—e.g., create a *.java file—you compile the code—e.g., creating a *.class file, which is written in bytecode, and when you execute it, an interpreter—the Java.EXE—is responsible for actually executing the

Learn how analytical data
can help you make
informed business decisions



[Get your Whitepaper here](#)

Contact Us



[Get Your Free E-Book Download](#)

commands written in the bytecode format within the *.class file. As with any interpreter, since we are not dealing with real object code, one can manipulate the actual code written in the executed file.



For example, let's say you want to add functionality to a Perl/PHP/JSP/ASP code—that's easy. You could simply open the file in a text editor, change the code, and next time it was executed it would behave differently. You could easily write a program that changes the code back and forth as you wish as a result of some user interface activity.

With bytecode it's the same concept, only a bit trickier. Try to open bytecode in a text editor—not something you want to work with...but still possible ☺. Anyhow, the way to manipulate the actual bytecode is by intervening during the class loading flow and changing code on the fly. Every JVM (Java Virtual Machine) will first load all the class files (sometime it will do it only when really required, but that doesn't change the following description) to its memory space, parsing the bytecode, and making it available for execution. The main() function, as it calls different classes, is actually accessing code which was prepared by the JVM's class loaders. There is a class loader hierarchy, and there is the issue of the classpath but all that is out of the scope of this post...So the basic concept of [bytecode instrumentation](#) is to add lines of bytecode before and after specific method calls within a class, and this can be done by intervening with the class loader. Back in the good old days, with JDK <1.5, you needed to really mess with the class loader code to do that. From JDK 1.5 and above, Java introduced the Java agent interface, which allows writing Java code that will be executed by the class loader itself, thus allowing the manipulation of the bytecode within every specific class, and making the whole process pretty straightforward to implement, thus the zillion different products for Java profiling and “transaction management” for Java applications.



Blog Topics

[APM](#)

[Application Performance Management](#)

[Application Performance Monitoring](#)

[Business Transaction Management](#)

[Capacity Planning](#)

[End-to-End Visibility](#)

[Information Security](#)

[Real User Monitoring](#)

Next up: What does bytecode instrumentation have to do with transaction tracing?

correlsense

Transaction Topology Mapping

🕒 Posted on November 16, 2015

[Apple iOS Encryption](#) →

[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](#)



November 2016

August 2016

July 2016

June 2016

May 2016

April 2016

March 2016

February 2016

January 2016

November 2015

October 2015

September 2015

August 2015

July 2015

June 2015

April 2015

March 2015

February 2015



September 2014
August 2014
July 2014
June 2014
May 2014
April 2014
March 2014
February 2014
January 2014
December 2013
November 2013
October 2013
September 2013
August 2013
July 2013
June 2013
May 2013
April 2013



March 2013

February 2013

January 2013

December 2012

November 2012

October 2012

September 2012

August 2012

July 2012

June 2012

May 2012

April 2012

March 2012

February 2012

January 2012

December 2011

November 2011

October 2011



September 2011
August 2011
July 2011
June 2011
May 2011
April 2011
March 2011
February 2011
January 2011
December 2010
November 2010
October 2010
September 2010
August 2010
July 2010
June 2010
May 2010
April 2010



March 2010
February 2010
January 2010
November 2009
October 2009
August 2009
July 2009
June 2009
May 2009
February 2009
September 2008
July 2008
June 2008
February 2008
January 2008
December 2007



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



The Correlsense website uses cookies. [Find out more in our Privacy Policy](#)

Okay, thanks