

Java Bytecode Instrumentation and Transaction Tracing



Imagine you want to debug your code, or better yet, profile your code during run time. Bytecode instrumentation (BCI) ([see my last post](#)) is a perfect solution, since by using BCI an external tool can add code to every beginning and ending of every method call within every class....thus allowing you to measure performance and gather method-related data (variable info and such). However, these performance metrics have to be stored somewhere, and have to be

sent somewhere as well so the developer can actually look at the output. Applying this kind of procedure to too many method calls will eventually cause the code to execute very slowly since for every method called, you need to gather and send the performance data. This slowness is called “overhead.” In other words, bytecode instrumentation adds overhead by definition.

Learn how analytical data
can help you make
informed business decisions



[Get your Whitepaper here](#)

Contact Us



Get Your Free E-Book Download



In order to create a “transaction trace” for a transaction arriving into a Java Virtual Machine (JVM), one can intercept using BCI all the method calls that are serving HTTP calls to create a unique key (based on session or not). Now this key can be passed (different techniques can do this) to every method that is being called by the “parent” method. The method metric is reported with this key to a repository, thus creating a potential “method trace” of the specific incoming HTTP request. This basic concept is what is used by the different Java-oriented products for transaction management.

In order to improve the above and adapt more to a production environment, you’ll want to reduce the full method trace to just incoming/outgoing calls and maybe a few in between. This cannot be accomplished generically, since if you are not passing the “trace key” between every method, you may have a problem following the trace uniquely. The solution is to create bytecode which is specific to the Java application you are trying to create a trace for.

Java application servers such as [WebSphere](#) or [WebLogic](#) (Oracle today) are at the end of the day just another Java application execute by a JVM. This is the exact reason a custom solution for transaction tracing has to be created for every version of a Java application server since classes and methods are changing their names and internal structure. Of course the same goes to a standalone JVM that is executing a Java application of some sort.

The bottom line is that you can either use a generic trace that will intercept all the classes and method calls and eliminate the ability to use it in a production environment, or use a more tailored implementation for a specific application version and try to create



Blog Topics

[APM](#)

[Application Performance Management](#)

[Application Performance Monitoring](#)

[Business Transaction Management](#)

[Capacity Planning](#)

[End-to-End Visibility](#)

[Information Security](#)

[Real User Monitoring](#)

a partial trace that will reduce the overhead. Of course, keep in mind that for every new type of application, you will need to create a new version of your BCI as well.



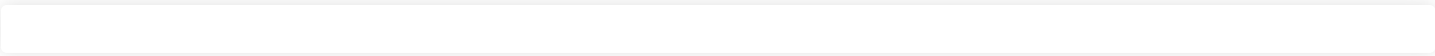
Next post: [More bytecode instrumentation limitations](#)

⌚ Posted on December 27, 2010

🔗 [bytecode instrumentation](#), [java](#), [Lanir Shacham](#), [transaction tracing](#)

← [CorrelSense Continues to Make Headway in Crowded BTM Space](#)

[End-to-End Transaction Monitoring](#) →



[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



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

Okay, thanks

