

# JXINSIGHT TRACES

**JXInsight Traces** provides flexible contextual instrumentation and measurement capabilities allowing for specific individual code blocks or high level business transactions to be profiled.

Both the contextual trace stack and trace path created during the execution of a business transaction augment the Java call stack recorded by providing additional information related to the execution context such as the JNDI name of a component, the SQL associated with a query or the URL for a HTTP request.

Contextual execution tracing is extremely important in understanding the execution flow patterns within an application.

Without prior knowledge of the execution patterns via the construction of a contextual software execution model that is devoid of system level concerns (concurrency, contention, co-ordination, and capacity) even the most experienced performance engineer will be lost in a sea of system related monitoring data.

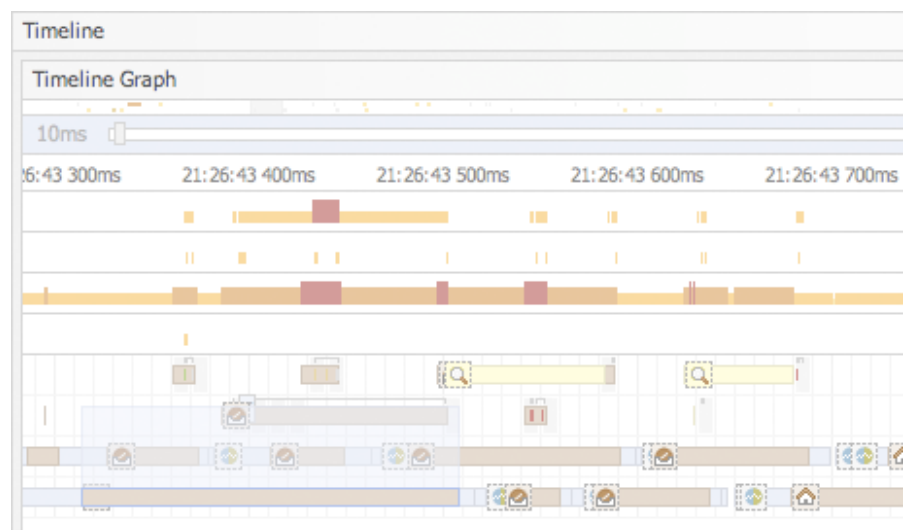
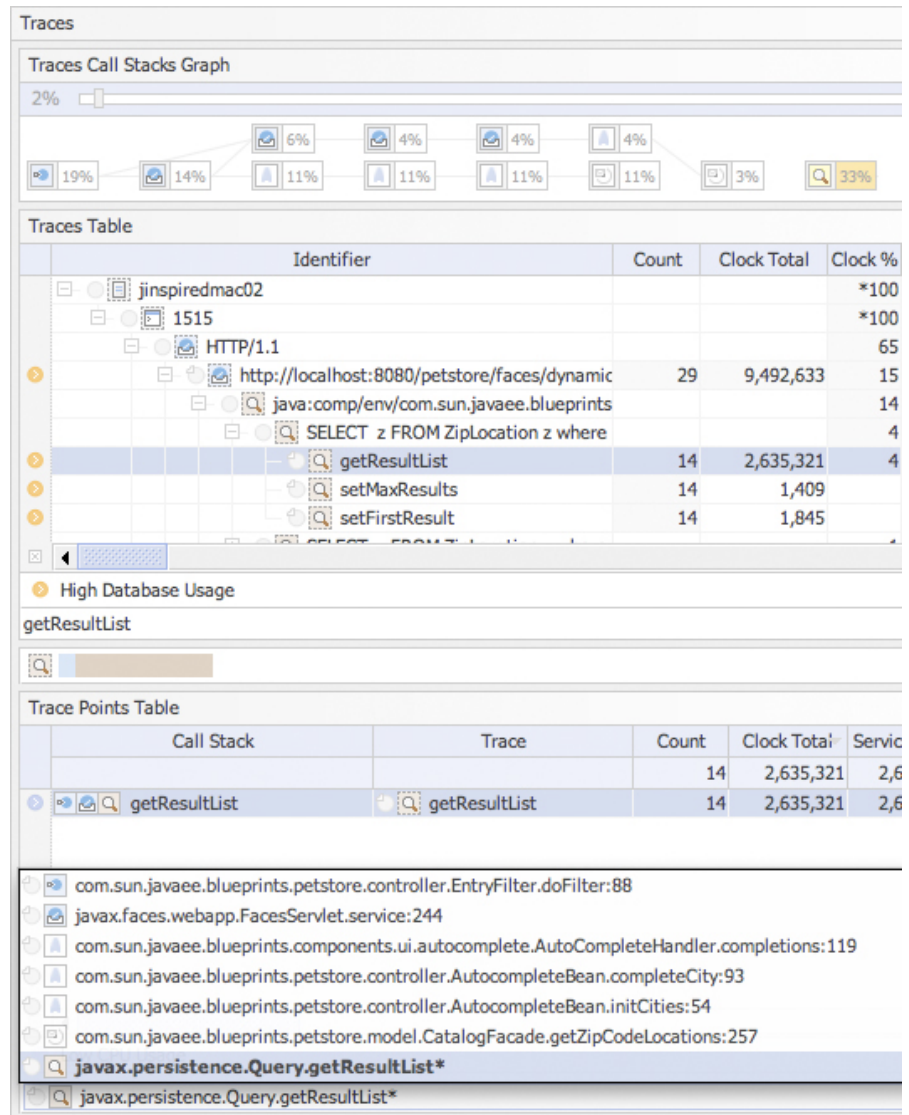
The software execution model helps developers to understand the runtime behavior of static software artifacts under construction. Its serves as a frame of reference for other software execution models such as system and runtime metrics as well as resource metering.

During development the improved insight into the execution can help avoid many common performance problems such as excessive client-to-server-to-database round trips.

## Distributed Tracing

Contextual trace paths can be nested and transferred between threads and across client and server processes allowing for distributed trace correlation and aggregation.

Trace extensions can work in environments consisting of different middleware and communication technologies including JMS, CORBA, RMI, HTTP as well as proprietary protocols.



## Agent Architecture

An unique feature of its architecture is the ability to trace both client and server JVM processes without having each individual process connected to a central management server. Importantly the correlation of distributed traces from client to server(s) can be performed offline.

With each JVM agent having a high degree of autonomy there is no single point of failure. Multiple management consoles and terminals can be run to routinely monitor, analyze, and store profiling snapshots. Snapshots recording for particular partitions can be replicated across consoles and terminals.

## Call Flow Visualized

To expose the underlying software execution model the application management and monitoring console makes the call origin (trace point) of the trace immediately obvious by displaying the subset of the call stack that has been classified.

With a sufficient level of class classification across the components and systems within a software it becomes relatively easy to understand the overall execution flow just by scanning a series of traces.

## Trace Group Metrics

JXInsight provides the ability to map individual trace identifiers to trace groups and then to map a trace group's statistics such as clock time, GC time, thread waiting time and blocking time to one or more metrics that can be sampled. This flexibility allows for standard and custom defined trace extensions to have summary measurement groupings suitable for quick a break down of performance costs across tiers, systems and components.

## Extension Packs

JXInsight comes with an incredible number of extension packs for all major technology standards, platforms, frameworks and products including: JMS, JDBC, JCR, JTS, JTA, JSP, Java Servlet, JSF, JCA, JAX-RPC, JAX-WS, JNDI, EJB, JPA, JMX, JavaSpaces, JINI, RMI, CORBA, JBoss Remoting, WebLogic RMI, Oracle Coherence, GridGain, IBM ObjectGrid, JBoss Seam, Google Guice, Spring, OSGi, WebLogic EventServer, Jetty (Comet), Apache Tomcat, Apache Struts, Apache Lucene, CommonJ, and IBM MQ.

Traces Table			
Type	Identifier		Clock Total
host	jinspired-w2000		*215,811,
process	1515		*215,811,
commonj.classname	ListenerHolder		204,276,
commonj.operation	run		99,973,
wlevs.classname	ScopedTransitionStreamImpl		99,967,
wlevs.operation	onEvent		39,170,
wlevs.classname	EPLProcessorImpl		39,170,
wlevs.operation	onEvent		19,688,
wlevs.classname	StageProxy		17,717,
wlevs.classname	EPLStatementImp		1,971,
wlevs.operation	sendEvent		
commonj.classname	CommonjWorkManagerImpl		6,
spring.classname	ServiceDependentOsgiBundleXmlApplicatic		10,194,
spring.classname	OsgiBundleXmlApplicationContext		479,
jndi.context.jndi	<default>		371,
commonj.classname	CommonjWorkManagerImpl		127,
wlevs.classname	LoadGenAdapterImpl		42,
wlevs.classname	EPLProcessorImpl		26,
spring.classname	BundleFactoryBean\$1		12,
wlevs.classname	ScopedTransitionStreamImpl		5,
commonj.classname	AcceptorThread		3,
commonj.classname	ActiveAdapter\$LoggingWorkListener		
commonj.classname	NativeAsyncPoller		
commonj.classname	NativeAsyncAcceptor		
commonj.classname	WorkManagerThreadPool\$1		
jndi.initialcontext	<init>		

Trace Points Table			
Call Stack	Trace	Count	Clock Total
		43,392	1,971,2
sendEvent:48	sendEvent	43,392	1,971,2