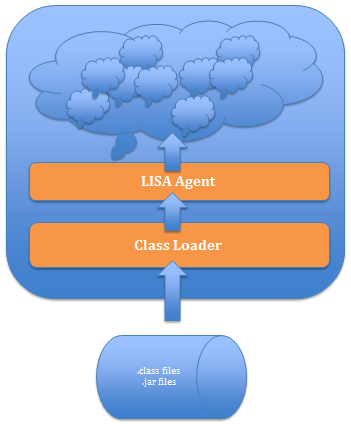
# Agent Overview

The LISA Java Agent is a piece of server-side technology that can be installed inside any Java process (including Java EE containers) and allows LISA to control and monitor server-side activities.

The agent can do what most profilers do (monitor loaded classes/objects, CPU usage, memory usage, threads, track method calls, and so on) but works across multiple JVMs and is used with LISA to bring unique features to the testing game.

In particular, it can give you visibility into what each test or even test step causes the server(s) to do behind the scenes so as to identify bugs and bottlenecks. This capability is similar to what LISA Pathfinder does, but works across all protocols used by Java applications without the need to instrument any code or even any configuration files.

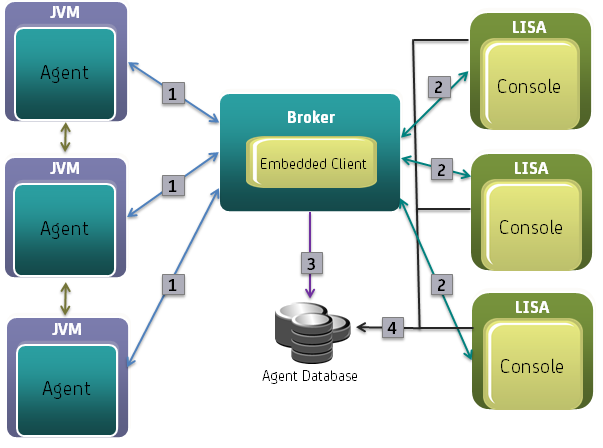
Another target area for the agent is **supporting LISA Virtual Services Environment**. By enabling record and replay of traffic and/or method calls across protocols, it gives LISA Virtual Services Environment complete control over any areas of the target application that users may want to virtualize, and provides a unified framework to accomplish this regardless of protocol



# Agent Architecture

The LISA Java Agent is composed of the following components:

* + - The agent itself, which runs embedded in a Java process
    - The JMS broker with an embedded JMS client
    - The consoles (or clients): get their finalized data from the broker (if recent) and the database (if older) for display and user interaction
    - The database



The flow of data is fully specified by the following JMS destinations:

The **lisa.agent.info** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This gives the broker and the consoles a view of which agents are currently online and what their basic properties are.

The **lisa.agent.port** topic is carried over connection 1, produced by the agents and consumed by the broker. This gives the broker a view of the connections currently active between multiple agents.

The **lisa.agent.api** topic is carried over connections 1 and 2, produced by the consoles and consumed (and replied to) by the agents. This allows the consoles to invoke agent APIs over JMS.

The **lisa.broker.api** topic is carried over connection 2, produced by the consoles and consumed (and replied to) by the broker. This allows the consoles to invoke broker APIs over JMS.

The **lisa.stats** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This gives the consoles an idea of what kind of load the agents are currently under and lets the broker persist those to the database.

The **lisa.vse** topic is carried over connections 1 and 2, produced by the agents and consumed by the consoles. When VSE is turned on, the consoles receive VSE frames (and reply to them in playback mode).

The **lisa.tx.partial** queue is carried over connection 1, produced by the agents and consumed by the broker. When an agent has captured a partial transaction (that is, all the frames that happen in its JVM), the agent sends it to the broker for assembly.

The **lisa.tx.full** topic is carried over connection 2, produced by the broker and consumed by the consoles. When the broker is done assembling partial transactions received over lisa.tx.partial, the broker sends the full transactions to the consoles.

The **lisa.tx.incomplete** topic is carried over connection 2, produced by the broker and consumed by the consoles. This topic is similar to lisa.tx.full, but is used for transactions that could not be fully completed within the allowed timeout.

**JDBC** connection 3 is used when the broker saves StatsFrame objects or fully assembled TransactionFrame objects.

**JDBC** connection 4 is used by the consoles to perform their queries for transactions or statistics that are no longer held in memory.

<http://www.itko.com/download/release/lisa_remote/arch.html>

# Agent Download and install

<https://support.itko.com/confluence/display/DOC70/Agent+Downloads>

<https://support.itko.com/confluence/display/DOC70/Using+the+Agent+Install+Assistant>

<http://www.itko.com/download/release/lisa_remote/>

# How to Start up

**url**: if you want to specifiy a non default broker connection string you set the url parameter, as in:

JAVA\_TOOL\_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=url=tcp://192.168.1.100:61616

This combines with starting the broker on that connection string with: java -jar LisaAgent.jar -broker tcp://192.168.1.100:61616

**name** : the Agent can be given a name using the name parameter. This helps with readbility (agents are displayed with that name in the UI) but also serves to identify processes across shutdowns and startups by keeping some persistent identifier:

JAVA\_TOOL\_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=name=myjboss

**token**: if you want to make access to the Agent secure, you can use a password in the environment variable using the token parameter, as in:

JAVA\_TOOL\_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=token=passw0rd

**jar**: if you want to put LisaAgent.jar and JavaBinder(.dll, .lib, .so) in different directories you can specifiy the location of the jar with:

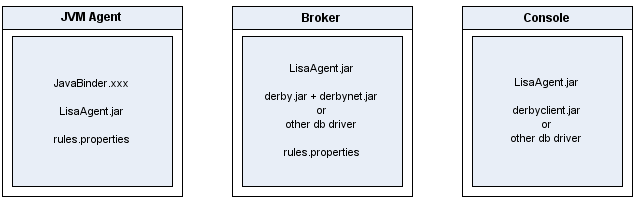
JAVA\_TOOL\_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=jar=file:<Path to LisaAgent.jar>

All options can be combined in any way you want by delimiting them with commas, e.g.:

JAVA\_TOOL\_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=url=tcp://orion:61616,name=myjboss,token=passw0rd

# Deployment

The required files for an agent deployment are illustrated below:

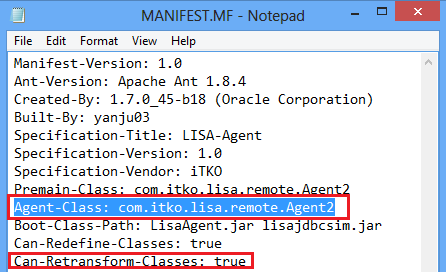
****

# –javaagent and JAVA\_TOOL\_OPTIONS

<http://docs.oracle.com/javase/6/docs/api/java/lang/instrument/package-summary.html>

-javaagent: *jarpath [=<options>]*

The manifest of agent JAR file must contain the attribute Premain-Class, the value of this attribute is the name of the *agent class*.



The agent class must implement a public static premain method below

*public static void premain(String agentArgs, Instrumentation inst)*

Provides services that allow Java agent will instrument programs running on the JVM. The mechanism for instrumentation is modification of byte-codes of motheds.

We can specify more than one java agent in command line. After the Java Virtual Machine (JVM) has initialized, each *premain* method will be called in the order the agents were specified, then real application main method will be called.

**What the mechanism for the argument ‘-javaagent’ in Lisa agent**:

Please check the source code Agent2.java : **L785**

if("main".equals(Thread.currentThread().getName())||"Main Thread".equals(Thread.currentThread().getName())) {

…

Agent.getInstance().**start**(m\_name, m\_token, m\_cn);

…

}

# TransactionDispatcher

main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\transactions\TransactionDispatcher.java

TransactionDispatcher.methodEnter(…)🡪getCurrentStack.pop🡪Create a new TransactionFrame object

TransactionDispatcher.methodExit(…)🡪sendTransactionInternal🡪sendMessage

# Pathfinder console

Source code: \main\portal\pathfinder\

Install GWT: <https://dl.google.com/eclipse/plugin/4.3>

Debug: <https://support.itko.com/confluence/display/DEV/Eclipse+GWT+Setup>

# Pathfinder Command Line

<https://support.itko.com/confluence/display/L7D/LISA+Pathfinder+Command-Line+Tool>

The **PFCmdLineTool** command lets you perform various LISA Pathfinder tasks from the command line.

The main options are --count, --roots, --paths, --export, --import, --baseline, --virtualize, and --extract-data.

This command has the following format:

**PFCmdLineTool** [--count|--roots|--paths|--export|--import|--baseline|--virtualize|--extract-data|--help|--version] [task-specific options] [search-criteria]

**Source code**: \main\pf-core-bridge\src\com\itko\lisa\pathfinder\PathfinderCmdLineTool.java

How to set weights for agent:

PFCmdLineTool.exe --set-weights --agent=TrainingTest --protocols=WebMethods,JMS,MQ --weights=8,8,8

# Dev Console

The options relevant to this section are:

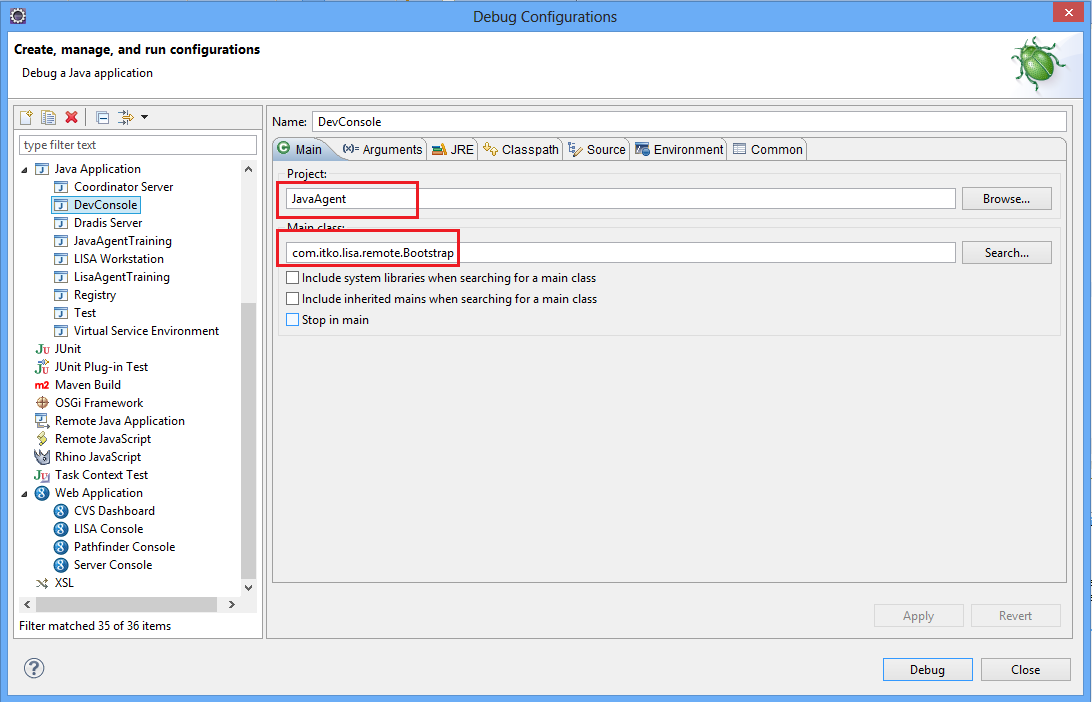
**~~java -jar LisaAgent.jar -console [connection url]~~** ~~to run the standalone console~~

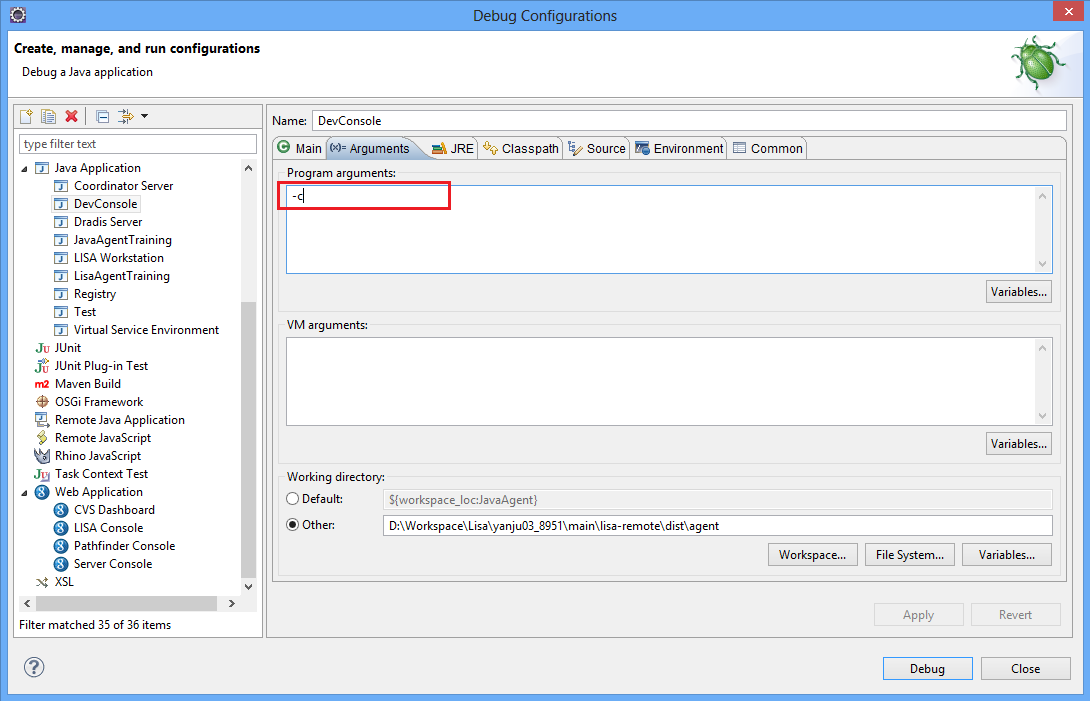
**~~java -jar LisaAgent.jar -dev [connection url]~~** ~~to run the console with an embedded broker~~

**We have separated the Dev console from LisaAgent2.jar**

**Java –jar DevConsole.java**

How to debug Dev console: (**Please make sure the registry is started before debug**)





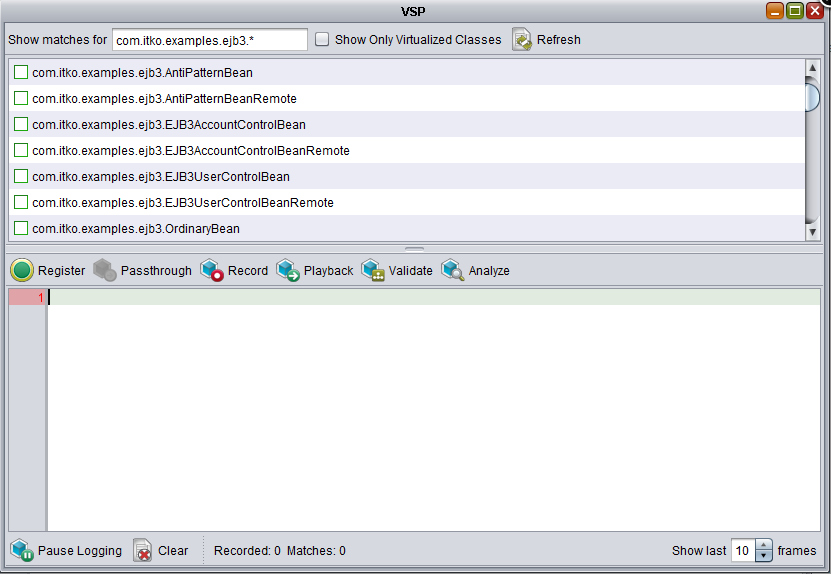
Start profiling: Only get the information from intercepted threads

Start Global profiling: Get all information from all threads, means capture everything

The VSP windows in Dev Console lets you perform the following tasks:

1. Dynamically change the behavior of Java methods
2. Obtain a list of suggested classes to virtualize
3. Make the Dev console act as a Virtual Service Environment(VSE) by clicking the ***Register*** button

<https://support.itko.com/confluence/display/DOC51/Using+the+Virtual+Service+Playground>



1. Register button : Make the Dev Console act as a Virtual Service Environment(VSE), you can see a VSE is available in Server Console after clicking this button
2. Passthrough : It corresponds to Live System execution mode. The VSE Passthrough provides a way to define the LISA VSE endpoints to the application and allow the VSE to first access the LIVE service. In the event the live system does not provide a suitable response, then the Virtual Service Image will be used.

<https://support.itko.com/confluence/display/technotes/Overview+of+Passthrough+VSE+to+a+Live+System>

1. Record button: The console receives recording events, if an agent is in VSE playback mode, the console will provides VSE responses to the agent.
2. Playback button: When you are in VSE playback mode, the console has nothing to response with because it does not have a virtual service mode and service image. However, if you record some transactions while the console is listening, it will create a trivial mode and image in memory (It is basically a map of responses keyed by request without any substitution or custom logic). You can use that as a trivial VSE to quickly test out some recording before you go through the whole VSE deployment cycle.
3. Validate button: This mode uses both the VSE and live system to derive a response to the current request. The responses are compared and appropriate history remembered. It allows comparison between the responses provided by VSE and a corresponding live system and, where differences exist, path or heal the VSE service image to keep in sync with the live system. This mode is also known as live healing mode. It is the latest efficient of all the modes, and is only available for Web Services.
4. Analyze button: The VSE Candidates window appears after clicking this button, the window contains the list of suggested classes. For each class, a score is provided.

**FID**: Unique identifier of the frame

**PID**: Unique identifier of the frame's parent

**TID**: Unique identifier of the transaction

**SID**: Unique identifier of the session

**CAT**: Category

**FLAGS**: Flags (if any)

**CX**: Complexity. The components of the value represent the frame depth, total, and categories.

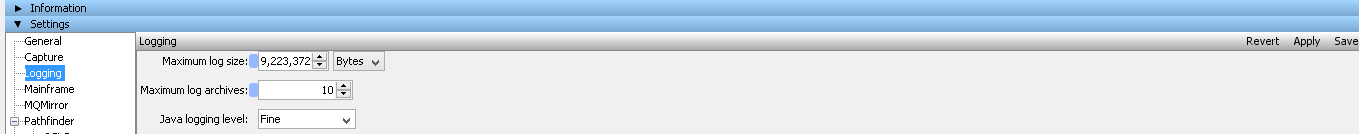
**SRC**: Source address (if applicable)

**TARGET**: Target address (if applicable)

**BAG**: A set of arbitrary, non-persistent properties set by agents on frames so they can be used by the broker to perform additional logic. Bags are currently used for secondary stitching (when there are load balancers).

# Pathfinder UI

1. Source code: \main\lisa-remote\PFUI\
2. Some tips:

a. Apply button:



b. Save button

i) Save agent:



ii) Save all changes:



The PFUI only save the setting changes to rules.xml on broker side.

How to get agent properties:

**public** Map<String, String> getAgentProperties(IAgentInfo info) **throws** JMSInvocationException

{

**if** (info.equals(**new** AgentInfo2())) **return** **new** HashMap<String, String>(0);

String code = **null**;

**if (info.isJava()) code = "return \_agent.getAgentProperties();";**

**if (info.isDotNet()) code = "return \_agent.AgentProperties;";**

JMSMethodInvocation jmi = **new** JMSMethodInvocation(code);

**return** (Map<String, String>) JMSConsoleClient.*invoke*(info, jmi);

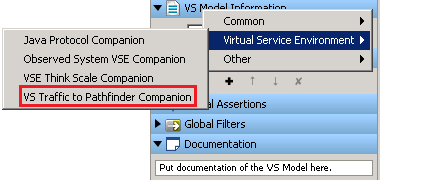
}

# **Pathfinder Companion**

The Pathfinder companion can be added in VS model, and it will capture the processed requests and responses and convert them as a **TransactionFrame** into PF database.

Source code: \main\pf-core-bridge\src\com\itko\lisa\pathfinder\PFDataFromVSCompanion.java

UI:



**Code Demo**:

<https://support.itko.com/confluence/download/attachments/28838622/Agent+Sidekick.arf?version=1&modificationDate=1382116221000>

# Developing against the Agent

**com.itko.lisa.remote.client.AgentClient**

**com.itko.lisa.remote.client.AgentClient**

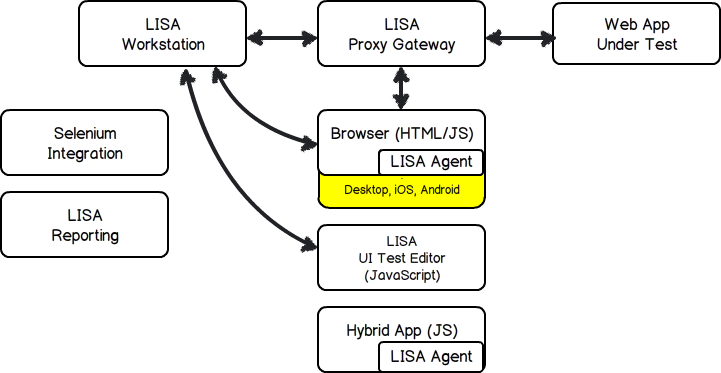
**com.itko.lisa.remote.client.DiscoveryClient**

**com.itko.lisa.remote.client.TransactionsClient**

**com.itko.lisa.remote.client.VSEClient.**

# Lisa Mobile UI

Source code: \main\lisa-mobile\



For more information:

<https://support.itko.com/confluence/display/main/Project+LISA+UI+Mobile>

# Competition

These products are not really competition because of the unique things that PF does, however these are the some products that might come up in a discussion while discussing PF

* + - Dynatrace
    - CA Wily introscope
    - New relic
    - Appdynamics

# Pathfinder Training

Session 1:

<https://catechnologies.webex.com/catechnologies/lsr.php?AT=pb&SP=MC&rID=50018732&rKey=2f962c7fe55f3ac1>

Session 2:

<https://catechnologies.webex.com/catechnologies/lsr.php?AT=pb&SP=MC&rID=50073562&rKey=ab8937fa309add4d>

Using Training.JAR Excersises to Understand JavaAgent Instrumentation：

https://support.itko.com/confluence/display/DEV/Using+Training.JAR+Excersises+to+Understand+JavaAgent+Instrumentation

# **Roadmap & Product enhancements**

* + New platforms to be added to the support Matrix:
    - WAS 8.0
    - WLC 9.2
    - Web methods 7/8
    - JBoss 5.1
  + Enable transaction topology map export capability
  + Enable stateful support for Baseline/VSE
  + Enable ACL for Pathfinder web console
  + Cognitive UI design
  + .NET prototype

<https://support.itko.com/confluence/pages/viewpage.action?pageId=28836527>

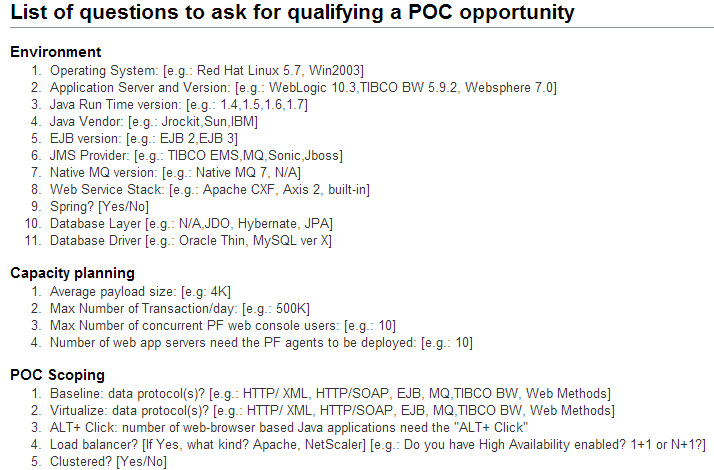
# **ITO – Project Discovery**

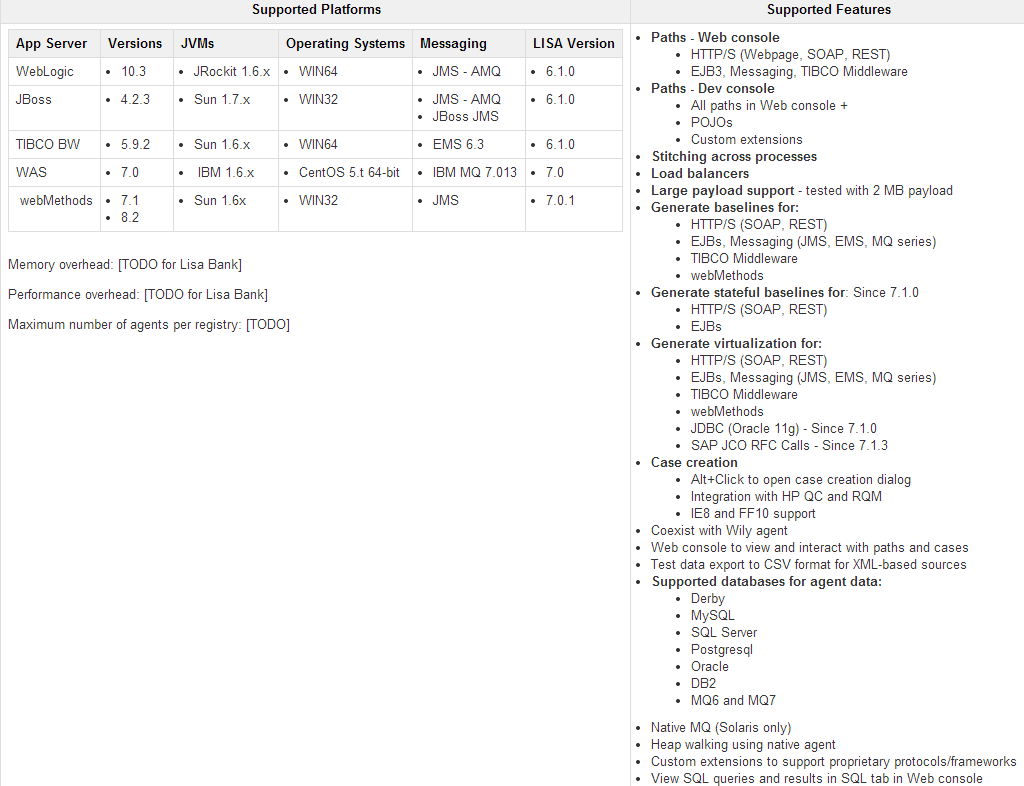
There are some great ideas in this ITO, for more details, please check:

<https://support.itko.com/confluence/display/DEV/Project+Discovery>

# **Supported Features**

<https://support.itko.com/confluence/display/DEV/Patfinder+POC+Checklist>





# **Java agent Troubleshooting**

<https://support.itko.com/confluence/display/DOC70/LISA+Java+Agent+Troubleshooting>

# Java Binder

In many environments the command line to start the application is not readily accessible. This often arises with applications that use embedded VMs (meaning they use the [JNI](http://baike.baidu.com/link?url=rojW83R9LWaLQbVzc_DUVUnoJOFe4WUa7O3U3fulocaQ5xTAd5Pus8HhYTZOAe9nTSJVOjbdHP8HvLkMxafJAa) Invocation API to start the VM), or where the startup is deeply nested in scripts. In these environments the JAVA\_TOOL\_OPTIONS environment variable can be useful to augment a command line.

When this environment variable is set, the JNI\_CreateJavaVM function (in the JNI Invocation API) prepends the value of the environment variable to the options supplied in its JavaVMInitArgs argument. In some cases this option is disabled for security reasons, for example, on Solaris OS the option is disabled when the effective user or group ID differs from the real ID.

This environment variable allows you to specify the initialization of tools, specifically the launching of native or Java programming language agents using the -agentlib or -javaagent options. In the following example the environment variable is set so that the HPROF profiler is launched when the application is started.

$ export JAVA\_TOOL\_OPTIONS="-agentlib:hprof"

This variable can also be used to augment the command line with other options for diagnostic purposes. For example, you can supply the -XX:OnError option to specify a script or command to be executed when a fatal error occurs.

Since this environment variable is examined at the time that JNI\_CreateJavaVM is called, it cannot be used to augment the command line with options that would normally be handled by the launcher, for example, VM selection using the -client or the -server option.

The JAVA\_TOOL\_OPTIONS environment variable is fully described in the [JAVA\_TOOL\_OPTIONS section of the JVM Tool Interface documentation](http://download.oracle.com/javase/7/docs/platform/jvmti/jvmti.html#tooloptions).

[**http://www.ibm.com/developerworks/cn/java/j-lo-jvmti/**](http://www.ibm.com/developerworks/cn/java/j-lo-jvmti/) **(Java thread monitor)**

[**http://docs.oracle.com/javase/7/docs/webnotes/tsg/TSG-VM/html/envvars.html**](http://docs.oracle.com/javase/7/docs/webnotes/tsg/TSG-VM/html/envvars.html)

**JVMTI**: Java Virtual Machine Tool Interface

**JVMDI**: Java Virtual Machine Debug Interface

**JVMPI**: Java Virtual Machine Profiling Interface

**Source code**: \main\lisa-remote\JavaBinder\

JVMTI: Java Virtual Machine Tool Interface

<http://docs.oracle.com/javase/7/docs/platform/jvmti/jvmti.html#jvmtiEvent>

For the project JavaBinder, We can support JVMTI, [JVMDI](http://docs.oracle.com/javase/1.5.0/docs/guide/jpda/jvmdi-spec.html) and [JVMPI](http://www.oracle.com/technetwork/articles/javase/jvmpitransition-138768.html).

Binder.cpp: Implement some interfaces of JVMTI

LegacyBinder.cpp: Implement some interfaces of JVMDI and JVMPI

# Transaction and Transaction Frame

Transactions and transaction frames are key concepts in LISA Pathfinder.

A transaction frame encapsulates data about a method call that the LISA Java Agent intercepted.

The data includes such information as:

* The name of the method
* The name of the class to which the method belongs
* The arguments that were passed to the method
* The value that the method returned
* Each transaction frame has a unique identifier.

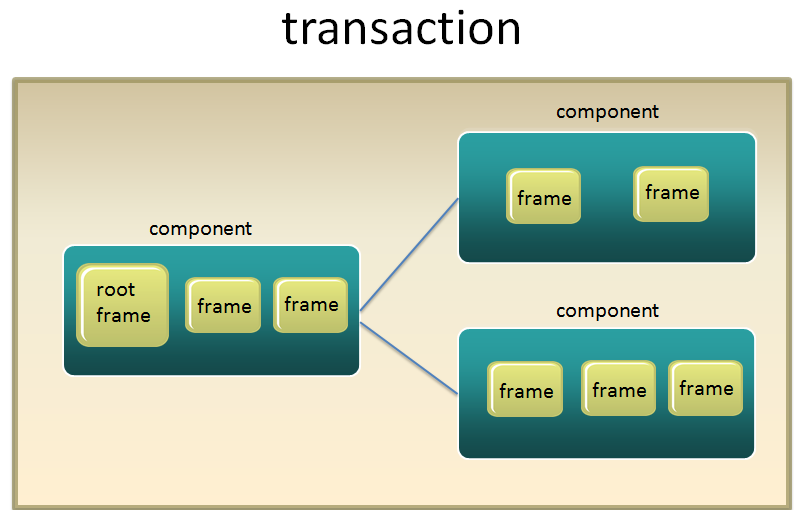
Each transaction frame has a category. Typically, the category represents a protocol or platform. Examples include EJB, JDBC, JMS, TIBCO, and WebSphere MQ.

A transaction is a code path that is executed by one or more servers as the result of a client request. Each transaction contains a hierarchical set of transaction frames. The top-level frame in the hierarchy is referred to as the root transaction frame. Each transaction has a unique identifier.

Note: In the LISA Pathfinder Console, transactions are referred to as paths.

The following graphic shows the relationship between transactions and transaction frames. The transaction frames are grouped into components. A component can be thought of as a service boundary within the code path of a transaction.

Diagram shows a transaction and its transaction frames.



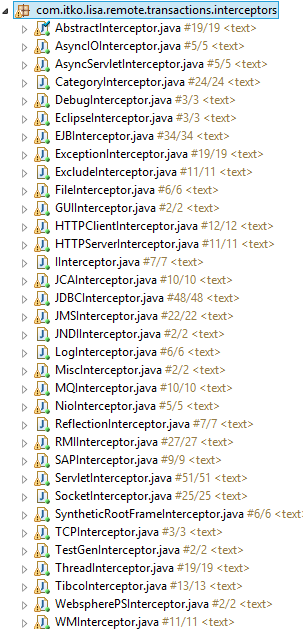
# Interceptor and Dispatcher

## Interceptors

Interceptors are the first layer of filtering of methods. Interceptors have rules that define which method of which class cary some interest and should be hijacked. The rule also define which method call or set of methods should be handled by which dispatcher. loadRules method is responsible for defining custom rules.

block method defines which methods or classes should be skipped early, to avoid any mess in preProcess and postProcess.

preProcess and postProcess are the hook-up methods called before and after method body. The implementations usually filter out some cases to reduce the noise.



## Dispatchers

Dispatchers - components that have methodEnter and methodExit methods. It decide itself whether to skip handling the method, or handle it. For example FileDispatcher is interested only in FileInputStrean and FileOutputStream and RandomAccessFile - it will only handle methods of those three, but skip all other.

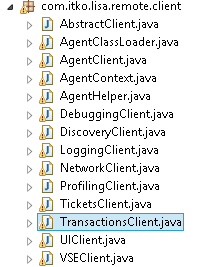
# Test Step

## Subprocess

Subprocess can be used as module in other test cases, which increase their ability to be reused. You can build a library of subprocesses that can be shared across many test cases.

<https://support.itko.com/confluence/display/DOC71/Building+Subprocesses>

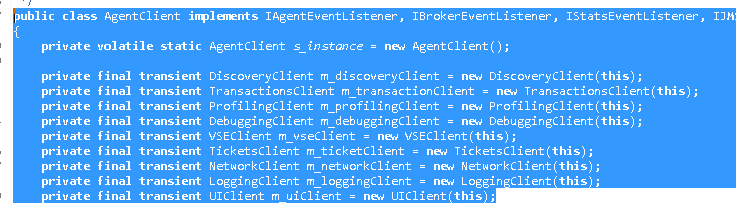
# Clients



How to get the client instance?

Step1: AgentClient.getInstance();

Step2: AgentClient.getxxx();



# DevConsole(ATK) – Live Path

How the ATK gets the live path?

Step1: JMSAgentClient. sendTransactionInternal(TransactionFrame)

Step2: JMSBrokerClient.onMessage(..), the broker received the partial transactions from agent

Step3: JMSBrokerClient.sendFullTransaction(root).

//Send a full transaction (JMSConnectionConstants.FULL\_TX\_TOPIC) to the consoles after it has been assembled

So it means the DevConsole(ATK) always receive the assembled transaction(s)

# Pathfinder Performance

<https://support.itko.com/confluence/display/DEV/Pathfinder+Performance>

# C:\Users\yanju03.TANT-A01\AppData\Local\YNote\Data\yanjunnf@gmail.com\fc6805c5631e48e49ed3729bcd7a03ce\clipboard.pngHow to transform class content (byte array)

1. Load all interceptor class, ***TransactionDispatcher.java : L139***



1. Transform all loaded classes

Agent2.java.transform(….)

|-->RulesInternal.*exclude(…)*

|-->Start agent (**L785**)

|-->m\_cold.transform(…) //ColdSwapper.java

|--> check ***s\_transformDepth //L785***

|--> RulesInternal.exclude(..)

|--> if (Rules.hasExclude(ccpackage, "\*", "\*") && !Rules.hasInclude(ccpackage, "\*", "\*")) //These rules are defined in Rules.properties **L977**

|-->if(Rules.hasFullExclude(cc.getName())&&!Rules.hasFullInclude(cc.getName()) && !Rules.hasInclude(ccpackage, "\*", "\*") && !Rules.hasInclude(cc.getName(), "\*", "\*")) //These rules are defined in Rules.properties **L997**

|--> transformByHierarchy //ColdSwapper.java : **L1043**

|--> Set iMethods = new HashSet(); //Common method & Interface methods

|--> Set vMethods = new HashSet(); //Virtualized methods

|--> Map sMethods = new HashMap(); //session methods

|--> **clone** = new HashMap(Rules.getIntercepts()); //L1313

|--> instrumentMethod(…) //ColdSwapper.java : L2732

***s***

# Rules.properties(Rules.java & RulesParser.java)

**RulesParser.java** : **private static void parseRule(String line) {**

**}**

Type:

public static final int TYPE\_MONITOR = 1; //method

public static final int TYPE\_VIRTUALIZE = 2; //Virtualization

public static final int TYPE\_SESSION = 4; //Session, *track=*

*Rules.java:*

*We can see the following 3 kinds of agent, the priority is:*

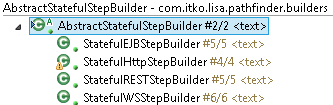
1. Standalone agent
2. Agent group
3. Default agent

# Pathfinder Console Code Analysis

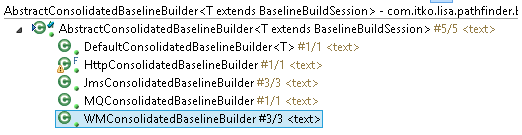
Build Session: It is used for building LISA model and related assets

1. BaselineBuildSession
2. VSAssetBuildSession
3. ExtractDataBuildSession

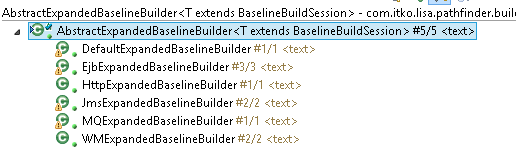
Seems like we only support create stateful baseline for EJB, Http, REST and Web Service (HTTP, HTTPS)



**ConsolidateBaseLineBuilder**:



**ExpandedBaselineBuilder**:



//main\pf-core-bridge\src\com\itko\lisa\pathfinder\builders\BuilderRegistry.java

Please reference BuilderRegistry::initialize(…)

# Transaction Frame Flag

1. FLAG\_INSTRUMENTED

A frame is instrumented if the network calls it makes or receives have been altered to inject tracking headers, it also means the head has been injected some properties to track later.

# Transaction Persist

1. Send transaction to broker, then add it into a buffer

TransactionArchival.bufferTransaction(TransactionFrame) line: 297 (out of synch)

TransactionAssembler.clientServerAssembly(TransactionFrame, Map<String,TransactionFrame>) line: 413 (out of synch)

TransactionAssembler.addPartialTransaction(TransactionFrame) line: 201 (out of synch)

JMSBrokerClient.onMessage(Message) line: 626 (out of synch)

JMSBrokerClient.access$000(Message) line: 77 (out of synch)

JMSBrokerClient$4.onMessage(Message) line: 326 (out of synch)

ActiveMQMessageConsumer.dispatch(MessageDispatch) line: 1229

ActiveMQSessionExecutor.dispatch(MessageDispatch) line: 315 (out of synch)

ActiveMQSessionExecutor.iterate() line: 461 (out of synch)

PooledTaskRunner.runTask() line: 122

PooledTaskRunner$1.run() line: 43

ThreadPoolExecutor.runWorker(ThreadPoolExecutor$Worker) line: 1145

ThreadPoolExecutor$Worker.run() line: 615

Thread.run() line: 744

1. Persist transaction frame
2. Way 1

com.itko.lisa.remote.plumbing.assembly.**TransactionAssembler.expiryLoop**()

com.itko.lisa.remote.plumbing.assembly.TransactionAssembler.{...}.new Thread() {...}.run()

1. Way 2

TransactionPersister.persistAllTransactions(List) line: 103 (out of synch)

TransactionPersister.persistTransactions(List) line: 95 (out of synch)

PersistManager.persistTransactions(List) line: 245 (out of synch)

TransactionArchival$1.run() line: 239 (out of synch)

1. The default assemble time is 30s

**public** **static** **final** WiredLong ***transaction\_assembly\_expiry*** = *wireLong*("lisa.broker.transaction.assembly.expiry", 30000L, ***BROKER*** | ***VITAL***, "transactions", "The maximum amount of time after which partial transactions are orphaned");

# Transaction Frame Noise

//Please reference source code:

***JavaAgent\src\com\itko\lisa\remote\client\utils\ServiceBoundaryTransactionFrameFilter.java***

**private** **boolean** compressUnneededLayers(TransactionFrame frame)

{

// Eliminate all 0 calls as noise.

**if** (frame.getParent().getParent() != **null** && eliminateByCategory(frame, TransactionFrame.*CATEGORY\_DEFAULT*))

**return** **false**;

// Set it aside if the frame is a log message frame.

**if** (setAsideLogFrame(frame))

**return** **false**;

// Collapse same-category children (just call stacks).

**if** (collapseSingleCategoryChildren(frame))

**return** **false**;

// PATH-167: Collapse duplicate JMS frames.

**if** (collapseDuplicateJMSFrames(frame, **false**))

**return** **false**;

// Ignore single RMI frames.

**if** (eliminateRMISingleFrame(frame))

**return** **false**;

// Ignore EJB machinery calls.

**if** (eliminateEJBMachineryFrame(frame))

**return** **false**;

// Eliminate RMI between this root and an EJB child.

**if** (eliminateRMIToEJB(frame))

**return** **false**;

// DB Compression.

**if** (collapseDBCallStack(frame))

**return** **false**;

// If a child is an exception, set it aside.

**if** (setAsideExceptionFrame(frame))

**return** **false**;

// We don't need framework frames either.

**if** (eliminateByCategory(frame, TransactionFrame.*CATEGORY\_FRAMEWORK*))

**return** **false**;

**if** (collapseHTTPOfREST(frame))

**return** **false**;

**if** (collapseWebsphereProcessServer(frame))

**return** **false**;

// Collapse EJB-WS-EJB to eliminate the first EJB plumbing call.

**if** (eliminateEJBProxy(frame))

**return** **false**;

// Also we need to collapse WS-EJB calls when the class name and method

// name are the same.

**if** (eliminateChildJavaOfWSWrapper(frame))

**return** **false**;

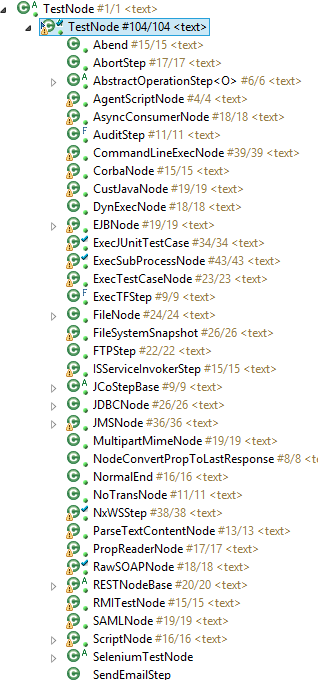
compressUnneededChildLayers(frame);

**return** **true**;

}

# Transaction Frame Test Step

Source code: main\pf-core-bridge\src\com\itko\lisa\agent\steps\ExecTFStep.java



# QA

**Q1: During creating baseline, what is used to do for the selected component?**

Answer: The selected component is a reference node.

For stateful baseline: its session is a key parameter, component’s name and category

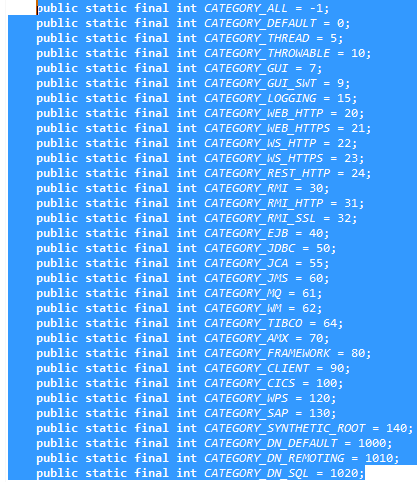
For expanded baseline or consolidated baseline, component’ name and category are key parameter,

main\pf-core-bridge\src\com\itko\lisa\pathfinder\builders\AbstractBuilder.java

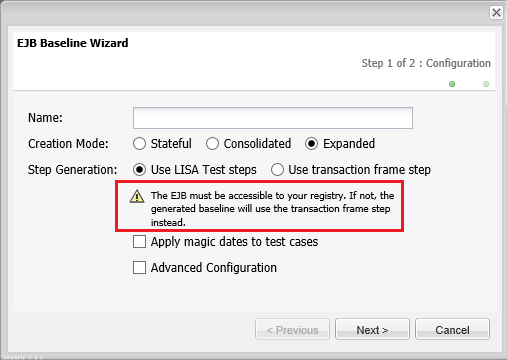
*AbstractBuilder.java::isRelevant(…)*

**Q2: Component Category***:*

*\main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\transactions\TransactionFrame.java*

**

**Q3: What is the exact meaning of the EBJ warning? Like below screenshot**

**

The following is the call stack:

EjbExpandedBaselineBuilder.createStep(TestCase, TransactionFrame, String) line: 114

EjbExpandedBaselineBuilder(AbstractExpandedBaselineBuilder<T>).createTest(PfiFile, TransactionFrame, List<String>) line: 120

EjbExpandedBaselineBuilder(AbstractExpandedBaselineBuilder<T>).buildBaselineAssets(PfiFile) line: 89

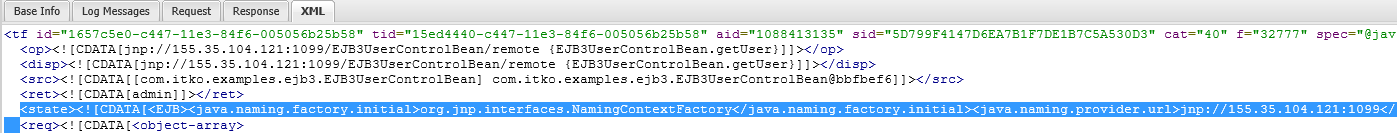
EjbExpandedBaselineBuilder(AbstractBaselineBuilder<T>).buildAssets(PfiFile) line: 106

PathfinderHelper.buildPfiFile(BuildSession) line: 394

PathfinderHelper.createPfiFile(BuildSession, TransactionFrame, List<TransactionFrame>, BuilderType, boolean) line: 325

First, Pay attention to the EjbExpandedBaselineBuilder::createStep(), If failed to create baseline with Lisa step, we will use transaction frame instead(Do it in *exception-catch* area).

Generally, when we capture a transaction frame of EJB, we can see some information about JNDI in state, just like:

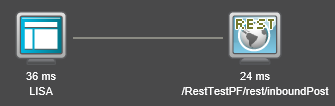


So we need to get the information from JNDI during creating test step of EJB. However, the JNDI provider always is <http://Registry:1099(The> registry has JNDI sever), so it means the EJB has accessible to the registry.

**Q4: How to setup JBOSS 6.3 on eclipse**

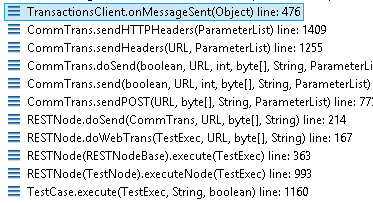
* Go to *Help > Install New Software*.
* Set *Work with* to the following URL depending on Eclipse version:
  + 3.6 (Helios): <http://download.jboss.org/jbosstools/updates/stable/helios/>
  + 3.7 (Indigo): <http://download.jboss.org/jbosstools/updates/stable/indigo/>
  + 4.2 (Juno): <http://download.jboss.org/jbosstools/updates/stable/juno/>
  + 4.3 (Kepler): <http://download.jboss.org/jbosstools/updates/stable/kepler/>
* In the results, make sure that you've *Group items by catagory* checked and then select *Web and Java EE development* somewhere near the bottom.
* *Next*, etc, *Finish*.

**Q5: How to combine two transaction frames from different agents to a transaction?**



For example, Agent1 send a rest API to agent2, then a transaction generated like above. First we need to add parent frame id into request,





Then the child frame can get the parent frame id.

**Q5: What kinds of Frame (categories) can be virtualized?**

* + Web service
  + Rest
  + CICS
  + IBM MQ
  + JMS
  + JDBC
  + Web method

For more details, please see:

\main\portal\pathfinder\src\com\itko\lisa\pathfinder\server\console\ConvertUtil.java

isVirtualizable(…)

**Q6: The call stack of execute test case of REST**

CommTrans.sendHTTPHeaders(ParameterList) line: 1409

CommTrans.sendHeaders(URL, ParameterList) line: 1255

CommTrans.doSend(boolean, URL, int, byte[], String, ParameterList) line: 955

CommTrans.send(boolean, URL, int, byte[], String, ParameterList) line: 795

CommTrans.sendPOST(URL, byte[], String, ParameterList) line: 773

RESTNode.doSend(CommTrans, URL, byte[], String) line: 214

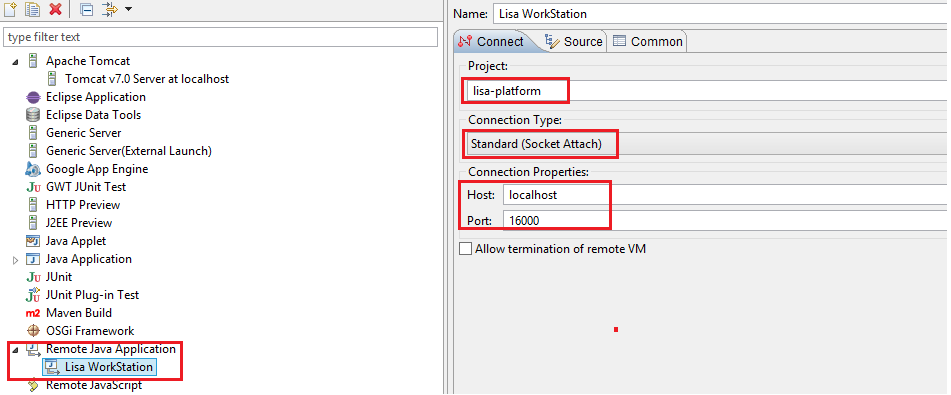
RESTNode.doWebTrans(TestExec, URL, byte[], String) line: 167

RESTNode(RESTNodeBase).execute(TestExec) line: 363

RESTNode(TestNode).executeNode(TestExec) line: 993

# Remote debug Lisa Workstation

<https://support.itko.com/confluence/display/~smorgan/Debugging+LISA>

1. Create a debug configuration like below:
2. 
3. Run ‘ant debug-lw’
4. Start ‘Lisa Workstation’ in eclipse

**Debugging Workstation from an Installation on Windows**

**cd \Lisa\bin**

**set LISA\_MORE\_VM\_PROPS=-Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=16000**

**lisa.bat com.itko.lisa.gui.TestManager**

**then connect to port 16000 with a remote debugger like Eclipse.**

**The same can be done for the registry, coordinator server, and simulator server, but make sure the debugging ports do not collide:**

**lisa.bat com.itko.lisa.coordinator.TestRegistryImpl**

**lisa.bat com.itko.lisa.coordinator.CoordinatorServerImpl**

**lisa.bat com.itko.lisa.coordinator.SimulatorServerImpl**

# How to stitch transaction frames on broker part

When agent send a transaction frame to broker:

TransactionAssembler.primaryServerAssembly() line: 299

TransactionAssembler.addPartialTransaction(TransactionFrame) line: 182

JMSBrokerClient.onMessage(Message) line: 626

The job of function ‘primaryServerAssembly’ is:

This is the method responsible for performing the actual building of the transaction from transaction fragments. The algorithm tries to match both upwards and downwards by looking up parent and children transactions held in temporary maps. If it can, it removes those fragments from the maps, replacing them with the newly assembled, larger fragments. If it can't, it just stores those fragments in the maps for later matching.

[Jun]**Search the transactions in buffer, look into any transaction frame is the parent of the incoming transaction frame’s root node, look into any transaction frame is the child of the incoming transaction frame’s leaf. The Condition is** *frameChild.getParentId = frameParent.getFrameId();*

We have two maps to store all transaction frames will be stored.

/\*A map of frame ids to frames for outgoing leaf nodes\*/

private static final Map<String, TransactionFrame> s\_***stagingAreaById*** = Collections.synchronizedMap(new HashMap<String, TransactionFrame>(1000));

/\*A map of parent ids to frames for incoming local root nodes\*/

private static final Map<String, TransactionFrame> s\_***stagingAreaByParentId*** = Collections.synchronizedMap(new HashMap<String, TransactionFrame>(1000));

These two maps above will be handled in the following function.

com.itko.lisa.remote.plumbing.assembly.**TransactionAssembler.expiryLoop**()

com.itko.lisa.remote.plumbing.assembly.TransactionAssembler.{...}.new **Thread**() {...}.run()

The method that gets run in an infinite loop to do **secondary assembly** and expiration of **timed-out** partial transactions

The most important function in ‘expiryLoop’ is **TransactionAssembly.secondaryServerAssembly**(..)

Before expiring partial frames, we give them a chance to assemble using secondary properties (instead of frameId/parentId). This is expected to be the case for TCP jumps across load balancers or appliances

TransactionFrame.FIELD\_TCP\_BYTES

TransactionFrame.FIELD\_TCP\_PAYLOAD

TransactionFrame.FIELD\_TCP\_PAYLOAD\_WEAK

TransactionFrame.FIELD\_CORRELATION\_ID

TransactionFrame.FIELD\_DESTINATION

For example, in the following graph, the parent node has two children, the parent may be the child for other transaction frame, the leaf may be the parent of other transactions

# Virtualization

1. VSE Protocol Manager:

* **new** VSEEJBManager()
* **new** VSERMIManager()
* **new** VSEJCAManager()
* **new** VSEJDBCManager()
* **new** VSETibcoManager()
* **new** VSEWMManager()
* **new** VSESAPManager()

**Source code**: main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\vse\VSEDispatcher.java

1. How and where store the virtualize information?  
   a. Add information during transformation.

* ColdSwapper

If we define some virtualized functions in rule.properties/rule.xml, we will handle it in ColdSwapper. Please check function ***ColdSwapper:: instrumentMethod(..)***, we can if one method is virtualized, we will call VSEDispatcher::methodHijack()

String playback = MessageFormat.*format*(playbackTemplate, **new** Object[] { **VSEDispatcher.class.getName(),** **Rules.*METHOD\_HIJACK***, src, className, cc.getName(), m.getName(), m.getSignature(), Rules.**class**.getName(), Rules.*NO\_HIJACK\_KEY*, newRet});

* HotSwapper

It is similar with above, the difference is this file will handle the runtime virtualized functions

All information of virtualized functions will be stored in:

**private** **static** **final** Map *s\_virtualizes* = **new** HashMap(); //com\itko\lisa\remote\instrument\Rules.java

b. During running

During running, the function VSEDispatcher:: methodHijack will be called, the call stack will be like below (For WM protocol):

VSEDispatcher::methodHijack()

🡪VSEWMManage::onHijack()

🡪 VSEFrame rframe = JMSAgentClient.sendVSEFrame(frame, true); //Send it to Broker

1. Do some description for step Java Listener Step.

The Virtual Java Listener Step is used to handle virtualized JVM calls, such as a call to an EJB or other remote system. It listens for method calls intercepted by the LISA agent, and converts them to a standard VSE request.

Source code: main\virtualize\src\com\itko\lisa\vse\stateful\protocol\java\listen\**JavaListenStep.java**

Let take a look at the initializing of this class

JavaListenStep::prepare(TestCase test)

🡪 JavaListenStep::registerClassesAndProtocols()

🡪 registerClassesOnAgent( iAgentInfo, playback );

🡪 DefaultAgentFacade::virtualize(…)

🡪VSEClient::virtualize(…)

🡪Send JMS Method Invocation to agent "\_agent.virtualize($1);

🡪…

🡪VSEDispatcher::virtualize(…)

🡪registerProtocolsOnAgent( iAgentInfo, playback );

🡪 DefaultAgentFacade:: virtualizeProtocol (…)

🡪VSEClient::virtualize(…)

🡪Send JMS Method Invocation to agent"\_agent.virtualizeProtocol ($1);

🡪…

🡪VSEDispatcher:: virtualizeProtocol (…)

1. What’s progress when deploy/stop a virtual service?

Deploy call stack:

HotSwapper.virtualizeMethod(CtMethod, String) line: 460

HotSwapper.virtualizeOneMethod(String, String, String, boolean) line: 430

VSEWMManager(VSEProtocolManager).virtualizeMethod(String, String, String, boolean) line: 191

VSEWMManager.virtualize(Map) line: 64

VSEDispatcher.virtualizeProtocol(String, Map) line: 504

Agent.virtualizeProtocol(String, Map) line: 7015

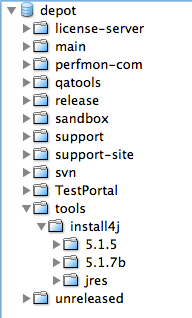
Un-deploy call stack:

VSEWMManager.unvirtualize(Map) line: 94

VSEDispatcher.unvirtualizeProtocol(String, Map) line: 520

Agent.unvirtualizeProtocol(String, Map) line: 7025

# Installer



//depot/tools/install4j/5.1.7b/... //your workspace/install/tools/install4j/...

//depot/tools/install4j/jres/... //your workspace/install/tools/install4j/jres/...

# JMS

Thre

# JMS Invoke

//Paste some example code here, Send a message of method invoke from console to agent:

**public** String invoke(IAgentInfo info, TransactionFrame frame, **InvocationResolver** resolver) **throws** JMSInvocationException

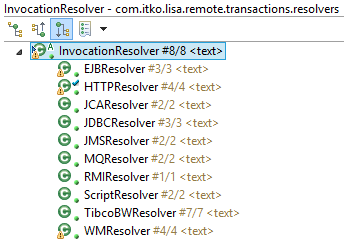
{

String code = "return $1.invoke($2);";

JMSMethodInvocation jmi = **new** **JMSMethodInvocation**(code, **new** Serializable[] { frame, resolver }, Math.*max*(10 \* Rules.*jms\_invoke\_timeout*.get() , 1200000L \* frame.getClockDuration()));

**return** (String) **JMSConsoleClient**.*invoke*(info, jmi);

}



HTTPResolver🡪HttpClient::request(…)

1. JMSConsoleClient
2. JMSAgentClient
3. JMSBrokerClient

# Agent Database schema

\main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\persist\SchemaPersister.java

\main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\persist\PersistManager.java

Some rules about agent database

1. *persistence\_disabled*
2. How to configure database information?

We can set properties in lisa.properties, just like below:

*lisadb.pool.common.driverClass=org.apache.derby.jdbc.ClientDriver*

*lisadb.pool.common.url=jdbc:derby://localhost:1528/database/lisa.db;create=true*

*lisadb.pool.common.user=rpt*

*lisadb.pool.common.password\_enc=76f271db3661fd50082e68d4b953fbee*

*lisadb.pool.common.minPoolSize=0*

*lisadb.pool.common.initialPoolSize=0*

*lisadb.pool.common.maxPoolSize=10*

*lisadb.pool.common.acquireIncrement=1*

*lisadb.pool.common.maxIdleTime=45*

*lisadb.pool.common.idleConnectionTestPeriod=5*

1. Materials:

<https://support.itko.com/confluence/display/technotes/Changing+LISA%27s+database+using+site.properties>

1. A customer how to build database schema:

In some cases they want the DDL to build the schema first. When this is the case, the best approach is to have the customer’s DBA perform the following actions:

1). Provide a temporary space for LISA to point to.

2). Configure site.properties to point to temp db space provided in Step 1.

3). Start the Lisa Registry.

4). Have customer DBA extract ddl from the temporary space.

5). Use ddl to create schema in production schema.

5.

# HTTPListenStep

The related technologies:

1. JAVA NIO <http://www.iteye.com/topic/834447> <http://blog.csdn.net/a9529lty/article/details/6454145>
2. Blocking Queue <http://www.cnblogs.com/jackyuj/archive/2010/11/24/1886553.html>

HttpListenStep::execute()

|--->new TextHTTPServer(…)==>|🡪EdPoint List //Make use of endpoint list to compare htpp request  
 |🡪NIOPortServer **extends** PortServer

NIOPortServer==> |🡪ServerSocketChannel *//This socket channel is used for ‘accept’ operation*

|🡪SelectorThread *//This thread is used for registering channels, put & pop blocking task*

SelectorThread==> |🡪Selector

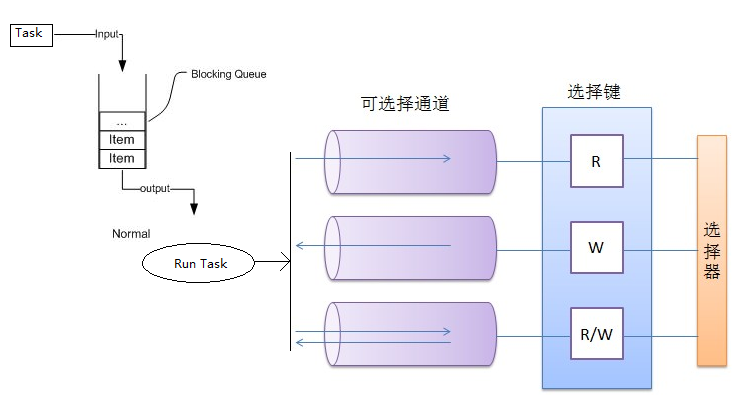
|🡪 LinkedBlockingQueue<**Runnable**> xxx

What is the procedure of accepting a client connection?

NIOPortServer::handleAccept()

|--->Get a channel to communicate with client, then call handleNewClientChannel(…)

|--->new PlainSession(…) or new SSLSession(…), we use these two class to handle client request



# HttpRespondStep

For VSE(such as REST category), the step ***HttpRespondStep*** return the response

HTTPTransaction.setResponseContent(String, String) line: 832

HttpRespondStep.setupResponseInfoHeaderAndContent(TestExec, HTTPTransaction, TransientResponse) line: 222

HttpRespondStep.respond(TestExec) line: 111

HttpRespondStep(BaseRespondStep).execute(TestExec) line: 231

HttpRespondStep(TestNode).executeNode(TestExec) line: 973

TestCase.execute(TestExec, String, boolean) line: 1230

TestCase.execute(TestExec, String) line: 1145

TestCase.executeNextNode(TestExec) line: 1130

TestCase.executeTest(TestExec) line: 1074

# Javaassit library

We use this library to inject code automatically

<http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/>

<http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/tutorial/tutorial.html>

<http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/>

<http://zhxing.iteye.com/blog/1703305>

|  |  |
| --- | --- |
| $0, $1, $2, ... | this and actual parameters |
| $args | An array of parameters. The type of $args is Object[]. |
| $$ | All actual parameters.For example, m($$) is equivalent to m($1,$2,...) |
| $cflow(...) | cflow variable |
| $r | The result type. It is used in a cast expression. |
| $w | The wrapper type. It is used in a cast expression. |
| $\_ | The resulting value |
| $sig | An array of java.lang.Class objects representing the formal parameter types |
| $type | A java.lang.Class object representing the formal result type. |
| $class | A java.lang.Class object representing the class currently edited. |

# Change Agent log level

Agent log level:

**public** **static** **final** **int** ***ERROR*** = 1;

**public** **static** **final** **int** ***WARNING*** = 2;

**public** **static** **final** **int** ***INFO*** = 3;

**public** **static** **final** **int** ***DEBUG*** = 4;

**public** **static** **final** **int** ***DEV*** = 5;

depot\main\lisa-remote\JavaAgent\src\com\itko\lisa\remote\utils\Log.java



For example:

lisa.log.level=’dev’

lisa.log.level.***logName*** = ‘dev’

# Lisa.properties

After changing any property in lisa.properties, we have to restart the application (Lisa Workstation, Registry) to apply the changes.

# BeanShell

1. For each agent/broker, we have a global interpreter object

JavaAgent/src/com/itko/lisa/remote/plumging/JMSBrokerClient.java

**private** **static** **final** Interpreter s\_interpreter = **new** Interpreter();

s\_interpreter.eval("import com.itko.lisa.remote.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.client.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.capture.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.instrument.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.instrument.rules.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.persist.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.plumbing.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.plumbing.assembly.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.transactions.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.utils.\*;");

s\_interpreter.eval("import com.itko.lisa.remote.client.utils.\*;");

1. How to send a jms invoke message to agent side.

All agents/consoles has a producer/subscriber for the topic **AGENT\_API\_TOPIC**, so when a console send a JMS invoke message, then all agents will receive this message, but in this message, we put a key **AGENT\_PID\_PROP**, this key is the agent id which will be as a condition in onMessage() function.

# Rules.xml

**//s\_properties holds values parsed from the agent rules.xml + system properties**

**private static final Properties s\_properties = System.getProperties();**

**private static final Properties s\_brokerProperties = new Properties();//Broker properties**

**//Live agent properties, it is different from s\_wiredProperties. The s\_wiredProperties is the completed works.**

**private static final Map s\_agentProperties = new HashMap();**

**private** **static** **final** Map ***s\_agentRulesList*** = **new** HashMap(); **//Persisted agents properties**

**//Agent group properties,** **Individual agent persisted rules**

**private** **static** **final** Map ***s\_groupRulesList*** = **new** HashMap()**;**

**//All wired properties are defined in rules.xml, it contains all agent properties.**

**//The PathFinder UI use the wired properties to generate controls.  
 private static final Map s\_wiredProperties = new HashMap();**

Nodes:

1. In rules.xml, each **<agent></agent>** will corresponds one **AgentRuleType** object
2. Priority:

Agent rule > Group rule

1. Get persisted rules
2. Retrieve rules from broker

Rules.xml on broker side > Rules.xml on agent side

What is the procedure of loading rules.xml

1. Load rules.xml, and store data into **s\_groupRulesList**.

**AgentRulesType.<init>(String, String) line: 39**

**RulesXmlSAXHandler.processAgentTag(String, Attributes, Map) line: 167**

**RulesXmlSAXHandler.startElement(String, String, String, Attributes) line: 91**

**SAXParserImpl$JAXPSAXParser(AbstractSAXParser).startElement(QName, XMLAttributes, Augmentations) line: not available**

**XMLDocumentScannerImpl(XMLDocumentFragmentScannerImpl).scanStartElement() line: not available**

**XMLDocumentScannerImpl$ContentDriver(XMLDocumentFragmentScannerImpl$FragmentContentDriver).next() line: not available**

**XMLDocumentScannerImpl.next() line: not available**

**XMLDocumentScannerImpl(XMLDocumentFragmentScannerImpl).scanDocument(boolean) line: not available**

**XIncludeAwareParserConfiguration(XML11Configuration).parse(boolean) line: not available**

**XIncludeAwareParserConfiguration(XML11Configuration).parse(XMLInputSource) line: not available**

**SAXParserImpl$JAXPSAXParser(XMLParser).parse(XMLInputSource) line: not available**

**SAXParserImpl$JAXPSAXParser(AbstractSAXParser).parse(InputSource) line: not available**

**SAXParserImpl$JAXPSAXParser.parse(InputSource) line: not available**

**SAXParserImpl.parse(InputSource, DefaultHandler) line: not available**

**SAXParserImpl(SAXParser).parse(InputStream, DefaultHandler) line: not available**

**RulesParser.loadCustomRulesXml(File) line: 205**

**RulesParser.loadCustomRulesProperty() line: 126**

**Rules.loadRules() line: 475**

**Rules.<clinit>() line: 453**

**ColdSwapper.<clinit>() line: 60**

**Agent2.premain(String, Instrumentation) line: 153**

1. Load rules from **s\_groupRulesList** to **s\_agentProperties**

**AgentRulesType.<init>(String, String) line: 39**

**AgentRulesType.copy(boolean) line: 406**

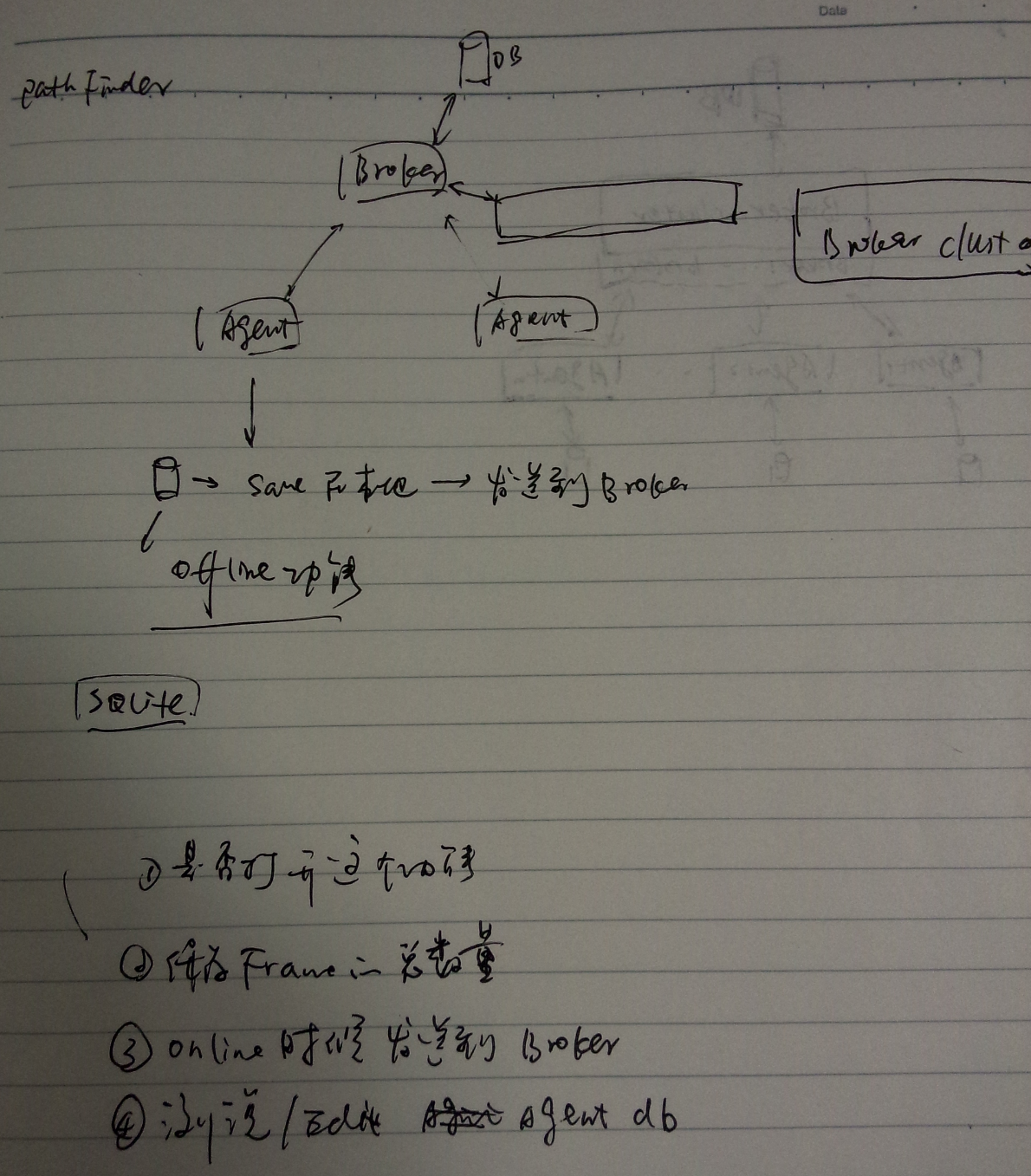
**RulesHandler.getAgentRules(long, boolean) line: 145**

**RulesHandler.loadAgentRules() line: 101**

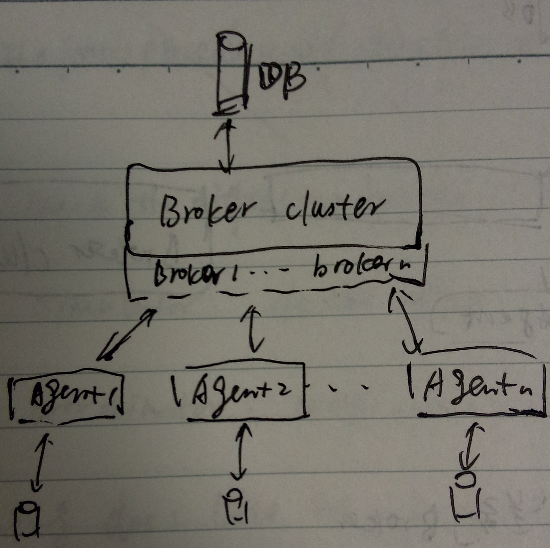
**Agent.start(String, String, String) line: 551**

# Something we can do

1. ~~Delete items from database~~ – ***Internal tool***
2. ~~Support store frames for office agent~~ –***ITO***
3. Go through source code
4. Interceptor class
5. Transaction Dispatcher class
6. Rule
7. AgentClient/DiscoverClient
8. VSE mode
9. Broker side



1. Broker Cluster



# Issues

1. After importing some transactions, the second time may fail.

[LISA AGENT:BC][ERROR][4168][715][qtp1328025401-715][May 24, 2014 8:04:20 PM] FindFrames exception: Syntax error: Encountered ")" at line 1, column 81.

1. Cannot delete frames for a specific agent
2. There are duplicate information when redeploy one virtual service.

HotSwapper.virtualizeMethod(…)

Put the following source code, it will generate class file.

{

//Write class to file

**try** {

**byte**[] classCode = cc.toBytecode();

String fileName = "C:\\LisaDump\\DumpStoreDir" + File.***separatorChar*** + className.replace('.', File.***separatorChar***) + ".class";

**int** pos = fileName.lastIndexOf(File.***separatorChar***);

**if** (pos > 0) {

String dir = fileName.substring(0, pos);

**if** (!dir.equals("."))

**new** File(dir).mkdirs();

}

FileOutputStream file = **new** FileOutputStream(fileName);

file.write(classCode);

file.close();

} **catch** (Exception e) {

}

}