Byteman: Tracing and Testing Made Easy
Andrew Dinn
Tuesday 8th March 2011
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
• Questions
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
• Questions
Why Trace? Why Test?

• We don't always know what our code is doing
  – not even with a debugger
    • impractical in many deployments
    • impractical with multi-threaded code

• We don't always know what our code might do
  – . . . in unusual circumstances
Get It Right First Time!

• Proving code is 'correct' is rarely an option

• Defining 'correctness' is tricky
  – implicit vs explicit definition
    • correctness proofs tend to want very explicit conditions
  – emergent understanding
    • proof refinement often means back to the drawing board
  – incomplete understanding
    • reliance on libraries and runtimes snookers us
  – and even if we can define it . . .

• Proving 'correctness' is usually intractable
  – I have done it twice in 25 years for select fragments of a larger system
So What Do We Actually Do?

• **We chip away at the problem**
  – unit test, integration test, system test, pilots, live monitoring

• **We write software to help see what our code is doing**
  – debug/product trace
  – execution stats collection
  – laborious, heavyweight and usually all or nothing

• **We write software to see what our code might do**
  – . . . in unusual circumstances
  – mock code, scaffolding, conditionally compiled builds
  – laborious, heavyweight and usually all or nothing

• **We test very different code to the released product**
  – . . . in very unusual circumstances
    • different code, different footprint, different timing
  – . . . invariably *not* the circumstances occurring in live install

• **We don't have 100% hindsight/foresight**
What Would We Prefer To Do?

• **Something much more flexible**
• **Highly selective, customisable and ad hoc tracing**
  – tweak code without needing to prepare source
  – at unit test, integration test, system test and in *live* deployments
  – use application and runtime data/functionality
  – revert back to original when done
    • needed for both live and multiple test deployments
• **Highly selective, customisable and ad hoc fault injection**
  – tweak code without needing to prepare source
  – at unit test, integration test and system test
    • in live deployments, anyone?
  – use application and runtime data/functionality
  – revert back to original when done
    • needed for multiple test deployments
AGENDA

- Why Trace? Why Test?
- How Does Byteman Help?
- How Do I Drive It?
- Questions
Byte (code) Man (ipulation)

• Available in a JVM near you right now
  – transform at load can redefine class structure and code
  – retransform after load can only redefine code
  – java.lang.instrument a pure byte bashing API

• Byteman makes it easy
  – inject actual Java code directly into Java code
    • direct manipulation
  – link to app/runtime code/data
    • what you say is what you get
    • type checking makes it safe
    • type inference keeps it simple

• Byteman makes it cheap
  – low transformation cost
  – tightly scoped changes

• Byteman makes it reversible
  – only ever redefines code
Example Byteman Rule

• Scripting Language
  – Simple, minimal structure for injected code
  – Event Condition Action Rules
  – Very Java-oriented
    • in fact it is Java, mostly!

RULE trace inactive transaction at commit
CLASS TransactionImple
METHOD commit()
AT ENTRY
BIND status : int = $0.getStatus()
IF status != javax.transaction.Status.STATUS_ACTIVE
DO traceStack("inactive commit " + $this +
               " status=" + status, 15);
ENDRULE
• Why Trace? Why Test?
• How Does Byteman Help?
  – Byteman Rule Language
• How Do I Drive It?
• Questions
E(B)CA Rules

• Event
  – CLASS/INTERFACE METHOD AT...
    • defines trigger point(s) i.e. location(s) in the code base
    • package, signature, return type are optional

• (BINDING)
  – introduces and initializes rule variables

• CONDITION
  – any Java boolean expression

• ACTION
  – any Java expressions

• Dynamically linked and typed
  – $0$ is the target of the trigger method, commit
  – getStatus is a method of TransactionImple
  – STATUS_ACTIVE references a static field of type int
Example Byteman Rule (2)

RULE simulate exception from Executor
INTERFACE ^java.util.Executor
METHOD execute
AT ENTRY
IF callerEquals("ServiceInstanceImpl.execute", true)
DO traceln("Throwing exception in execute");
   THROW new
      java.util.concurrent.RejectedExecutionException();
ENDRULE

• inject through the interface into implementors
• inject down into overriding implementations
  – AbstractExecutor implements Executor
  – ThreadPoolExecutor extends AbstractExecutor
• THROW/RETURN from trigger method call
  – must conform to method contract
  – bypass catch block processing (short-circuit)
Location Clauses

AT ENTRY
AT EXIT
AT/AFTER READ [[package.]type.]field | $localvar [count]
AT/AFTER WRITE [[package.]type.]field | $localvar [count]
AT/AFTER CALL [[package.]type.]method [(Types)] [count]
AT THROW [count]
AT LINE number

public check(Sym sym) throws BadSym, BadType
{
  String s = "";              // AFTER WRITE $s
  if (badSym(sym)) {
    // AT READ name 1
    s = munge(sym.name);      // AT CALL munge, AT WRITE $s 2
    throw new BadSym(s);      // AT THROW ALL
  } else if (badType(sym.type)) {
    // AT READ Type.name 1
    // AT CALL munge 2
    s = munge(sym.type.name); // AT CALL munge(TypeName) 1
    // AFTER WRITE $s 3
    throw new BadType(s);     // AT THROW 2
  }
}
Expressions

• Parameter, local and rule variables
  – $0, $1($this, $sym), $loopvar, status

• Special variables
  – $*, $# trigger method parameter array and parameter count
  – $! stacked return value in AT EXIT or AFTER CALL rule
  – @$ stacked arguments in AT CALL rule
  – $^ stacked throwable in AT THROW rule

• The full set of Java operations
  – operators +-*|, && ||, == < >, new, =, etc
  – instance/static field accesses and method invocations
  – built-in methods (any call with no target instance)
    – no control structures

• Assigning $ vars changes trigger method state
  – $1 = "Andrew"
  – $loopvar = $loopvar + 1
  – $! = 3
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
  – Byteman Built-In Methods
• How Do I Drive It?
• Questions
Built-in Methods

• **Tracing**
  – `traceOpen`, `traceClose`, `traceIn`, `traceStack`, ...

• **Managing Shared Rule State**
  – `flag`, `clear`, `countDown`, `incrementCounter`, ...

• **Timing**
  – `createTimer`, `getElapsedTime`, `resetTimer`

• **Checking Caller Stack**
  – `callerEquals`, `callerMatches`

• **Thread Synchronization**
  – `waitFor`, `signalWake`, `rendezvous`, `delay`

• **Recursive Trigger Management**
  – `setTriggering`
Example Byteman Rule (3.1)

- **XTS Coordinator Service**
  - negotiates 2 phase commit with remote Web Service Participants
  - sends PREPARE waits for PREPARED
  - logs participant details
  - sends COMMIT expects COMMITTED

- **XTS Crash Recovery Test**
  - kill JVM between logging and sending COMMIT then reboot
  - drop COMMITTED messages during first/second roll forward attempt
  - allow messages to pass and ensure TX completes at 3rd attempt

```plaintext
RULE drop committed message
CLASS CoordinatorEngine
METHOD committed(Notification, MAP, ArjunaContext)
AT ENTRY
BIND engine: CoordinatorEngine = $0,
   identifier: String = engine.getId()
IF getCountDown(identifier)
DO RETURN
ENDRULE
```
Example Byteman Rule (3.2)

RULE add coordinator engine countdown
CLASS CoordinatorEngine
METHOD <init>(String, boolean, EndpointReference, boolean, State)
AT EXIT
BIND engine:CoordinatorEngine = $0,
   identifier:String = engine.getId()
IF engine.recovered
DO createCountDown(identifier, 2)
ENDRULE

RULE countdown at commit
CLASS CoordinatorEngine
METHOD commit
AFTER WRITE status
BIND engine:CoordinatorEngine = $0
   identifier:String = engine.getId()
IF engine.recovered && countDown(identifier)
DO traceln("countdown completed for " + identifier)
ENDRULE
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
  – Rule Helpers
• How Do I Drive It?
• Questions
Helper Classes

- Built-ins are just public methods of a POJO
  - take a look
    - org.jboss.byteman.rule.Helper
- You can use any POJO as Helper

```java
class DBHelper {
    public void trace(String msg, Record rec) {
        // ...}
    
    RULE use my own trace method
    CLASS org.my.db.DBManager
    METHOD update(Record)
    AT CALL setName(String)
    HELPER org.my.bmutil.DBHelper
    IF $@[1] == "Andrew"
    DO trace("found interesting record update ", $1)
    ENDRULE
```
Helper Classes

• **HELPER clause outside rule resets for following rules**
  
  HELPER org.my.bmutil.DBHelper
  RULE my Helper rule 1
  . . .
  RULE my Helper rule 2
  . . .
  HELPER
  RULE back to default Helper
  . . .

• **Byteman type checks and links using named class**
  – Helper class must be in classpath
    • Rules injected into JVM code require helper class in bootstrap path
    • Byteman will install a jar into the bootstrap path if you ask

• **Often helps to extend Byteman Helper**
  class DBHelper extends Helper {
  . . .
  – allows you to reuse/redefine existing built-ins in your rules
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
• Questions
Four Different Routes

- **Java command line**
  - most complicated but applies rules from JVM start
    - intercept (almost) all JVM activity (e.g. inject into app Main())

- **Byteman bin shell scripts**
  - basic script just wraps up command line arguments
  - can install rules into an already running program (e.g. live JBoss AS)
  - can deinstall rules and reinstall
  - can also check status of loaded rules

- **Byteman API classes**
  - install the agent and install/uninstall rules from a Java program
  - doesn't have to be into the same JVM
  - used by contrib packages to do automatic rule loading/unloading

- **BMUnit package**
  - integration of Byteman into JUnit or TestNG
  - easiest way to load and unload Byteman rules
  - trivial to run from ant or maven
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
  – Java Command Line
• Questions
Java Command Line

• java option installs “Java agent” bytecode transformer
  -javaagent:/path/to/agent.jar=agentoptions

• Byteman main jar is a Java agent jar
  -javaagent:${BYTEMAN_HOME}/lib/byteman.jar=agentoptions
  • BYTEMAN_HOME is where you unzipped the download

• Byteman agent can start a listener on localhost:9090
  – allows upload/unload/reload/status of rules while program is running

• agentoptions are comma separated name:value pairs
  – e.g. script:/rules.btm,script:/morerule.btm,boot:byteman.jar

  script:script.btm       install rules from script.btm at agent startup
  boot:my.jar            add my.jar to bootstrap classpath
  sys:my.jar             add my.jar to system classpath
  listener:true          start up agent listener
  port:999               use listener port 999
  address:192.168.0.1     use listener host 192.168.0.1
  prop:name=value        configure Byteman System property

• where name is org.jboss.byteman.xxx
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
  – Byteman bin Shell Scripts
• Questions
bmjava

**bmjava javaargs**
- use in place of java command
  - **bmjava -cp build/classes Register -n Andrew**
    - installs Byteman agent, starts Byteman listener on localhost:9090
- bmjava options
  - these precede **javaargs**
    - **-p port -h hostname** use a different listener port/host
    - **-l /path/to/myscript.btm** load rules at agent startup
    - **-b /path/to/helper.jar** install jar into bootstrap path
    - **-s /path/to/helper.jar** install jar into sys path
    - **-Dorg.jboss.byteman.xxx** configure Byteman system properties

- rules injected as matching classes are loaded
- existing classes may need to be retransformed
  - e.g. **java.lang.Thread.start()**
bminstall

bminstall *procId* | *mainClass*

- installs Byteman agent into already running program
  - `bminstall -Dorg.jboss.byteman.debug org.jboss.Main`
  - always starts listener
    - `-p port` `-h hostname` use a different listener port/host
    - `-b` install byteman jar in boot path
      - should be the default (e.g. `bmjava.sh` provides `-nb`)
    - `-Dorg.jboss.byteman.transform.all` allow inject into `java.lang.*`
      - should be the default (e.g. `bmjava.sh` provides `-nj`
bmsubmit

bmsubmit [-l | -u] [script1 . . . scriptN]
  – load or unload rule scripts via Byteman listener
    • bmsubmit /path/to/myscript.btm
      – applies rules to new classes and retransforms existing classes
    • bmsubmit -u
      – removes rules and reverts affected classes
    • bmsubmit shows status of all loaded rules

-p port -h hostname    use a different listener port/host
-o outfile            redirect output to outfile

bmsubmit [-b | -s] jar1 [. . . jarN]
  – load jars into bootstrap or system classpath
    • bminstall -b /path/to/helper.jar

bmsubmit -c
  – list all loaded jars

bmsubmit -y
  – list current configured Byteman system properties
    • org.jboss.byteman.*
bmcheck

bmcheck [-cp path|jar]* [-p prefix]* script1 ... scriptN
  - parse and type check rules offline
    • bmcheck -cp my.jar -cp your.jar \n      -cp your.jar
      -p org.my -p org.your myscript.txt
  - needs to explicitly load classes mentioned in rules
    -cp locates jar containing classes mentioned in rules
    -p resolves unspecified packages in CLASS or INTERFACE clause
      • CLASS Foo ==> org.my.Foo
      • CLASS Bar ==> org.my.Bar, org.your.Bar
    • errors messages are now quite good and getting better
      - parser errors not always able to provide exact line
        • but usually close
      - type errors normally very precise
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
  – Byteman API Classes
• Questions
Byteman API Classes

- \texttt{org.jboss.byteman.agent.install.Install}
  - \texttt{main(String[])} used by \texttt{bmininstall}
  - other \texttt{static} methods for programs to use
    - \texttt{install(String pid, boolean addToBoot, String host, int port, String[] properties)}
    - \texttt{VMInfo[]} \texttt{availableVMS()}

- \texttt{org.jboss.byteman.agent.submit.Submit}
  - \texttt{main(String[])} used by \texttt{bmsubmit}
  - other \texttt{instance} methods for programs to use
    - \texttt{Submit()}
    - \texttt{Submit(String host, int port, PrintStream out)}
    - \texttt{addRulesFromFiles(List<String> filePaths)}
    - \texttt{addScripts(List<ScriptText> scripts)}

- \textbf{Used by contrib packages}
  - \texttt{dtest} instruments remote JVM for post-run validation
  - \texttt{BMUnit} integrates Byteman into JUnit and TestNG tests
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
  – BMUnit Package
• Questions
BMUnit Tests

• Integrates Byteman into JUnit and TestNG
  – automatically loads the agent for you
  – automatically loads and unloads rules for you

• Simply annotate your test classes and @Test methods
  – `@BMScript` identifies a script file to load
  – `@BMRule` provides rule text in the annotation
  – Class level annotation
    • load before running test methods, unload once all completed
  – Method level annotation
    • load before calling test method, unload after call completed

• JUnit: annotate test class with test runner
  – `@RunWith(BMUnitRunner.class)`
    class DBTests {
      ...

• TestNG: make test class extend runner
  – class DBTests extends BMNGRunner {
    ...

BMUnit Example

- package org.my.dbtests;
  @RunWith(BMUnitRunner.class)
  @BMScript(value="traceRules", dir="scripts")
  class DBTest1 {
    @Test
    @BMRule(className="FileOutputStream",
     methodName="<init>(File)",
     condition="$1.getName().contains("Andrew\")",
     action="THROW new FileNotFoundException()"
    )
    public void testDBFileHandler() { . . .
  }

- @BMScript name and/or dir can be defaulted
  - script dir dir defaults to test JVM's working directory
    - search for script file first in dir/org/my/dbtests then dir
  - script file name defaults from test class name and/or method name
    - DBTest1.btm class annotation
    - DBTest1-testDBFileHandler.btm method annotation
    - testDBFileHandler.btm method annotation
BMUnit From ant Or maven

• Execution just needs jars to be in the classpath
  
  ${BYTEMAN_HOME}/contrib/bmunit/byteman-bmunit.jar
  ${BYTEMAN_HOME}/lib/byteman-submit.jar
  ${BYTEMAN_HOME}/lib/byteman-install.jar
  ${BYTEMAN_HOME}/lib/byteman.jar
  ${JAVA_HOME}/lib/tools.jar

• For maven declare byteman jars as test dependencies
  – you'll find them in the JBoss repo (use 1.5.1+)
  – add tools.jar in your surefire configuration

    <configuration>
      <additionalClasspathElements>
        <additionalClasspathElement>
          ${java.home}/../lib/tools.jar
        </additionalClasspathElement>
      </additionalClasspathElements>
    </configuration>

• note the ../lib! maven points java.home at ${JAVA_HOME}/jre
Byteman Configuration Properties

- **-Dorg.jboss.byteman.debug**
  - enables printout from builtin method debug(String)
  - useful if you want to check your rules are actually firing
- **-Dorg.jboss.byteman.verbose**
  - enables agent internal tracing (also switches on debug)
  - lots of noise but you can see rules being injected and executed
  - let's you know when a rule is not being processed
- **-Dorg.jboss.byteman.transform.all**
  - enables injection into java.lang.* packages
  - requires `boot:/path/to/byteman.jar` or `bmininstall -b pid`
    - maybe also `boot:helper.jar` or `bmsubmit -b helper.jar`
- **-Dorg.jboss.byteman.compileTo bytecode**
  - injected code normally executed by interpreting parse tree
  - conversion to bytecode allows it to be JIT compiled
  - useful when rules are triggered frequently
  - currently applies to all rules but should be per-rule
AGENDA

• Why Trace? Why Test?
• How Does Byteman Help?
• How Do I Drive It?
• Questions
Questions

• **Byteman Project Page at JBoss**
  – http://www.jboss.org/byteman/

• **Downloads**
  – http://www.jboss.org/byteman/downloads
    • latest release 1.5.1.
    – also in JBoss maven repo (groupid: org.jboss.byteman)

• **Documentation**
  – http://www.jboss.org/byteman/documentation
    Programmers Guide (pdf)
  – contrib packages
    contrib/xxx/README.txt

• **User and Developer Forums**
  – follow link from project page

• **SVN Repository**
  – http://anonsvn.jboss.org/repos/byteman