What's your Java app up to?
Find out with Byteman
Andrew Dinn
Principal Software Engineer
Red Hat UK
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
• Driving Byteman
• Questions
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
• Driving Byteman
• Questions
Why Trace? Why Test?

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

• Don't always know what it might do
  – ... in unusual circumstances
Get It Right First Time!

• Proving code 'correct' rarely an option
• Defining 'correctness' is tricky
  – implicit vs explicit definition
    • correctness proofs tend to want the latter
  – emergent understanding
    • proof refinement usually means start again
• incomplete understanding
  • reliance on libraries and runtimes

• ... and often intractable
So What Do We Do?

• Chip away at the problem
  – unit, integration, system test
  – pilots, live monitoring

• Write code to see what code is doing
  – debug/product trace
  – execution stats collection
  – laborious, heavyweight, all or nothing
What Else Do We Do?

• Write code to see what code *might* do
  – . . . in unusual circumstances
  – mock code, scaffolding, conditional builds
  – laborious, heavyweight, all or nothing

• We test code very different to release
  – . . . in very unusual circumstances
  – different code, footprint, timing

• We do all this *without* 100% hindsight
What Would We Prefer To Do?

• Something Much More Flexible
  • Selective, customisable, ad hoc tracing
    – tweak without preparing source in advance
    – at unit, integration, system test & when live
    – use app/runtime data and functionality
    – revert back to original when done
  • needed for live and multi-test deployments
What Would We Prefer To Do?

• Selective, customisable, ad hoc fault injection
  – tweak without preparing source in advance
  – at unit, integration, system test (& live?)
  – use app/runtime data and functionality
  – revert back to original when done
• needed for multi-test deployments
Agenda

- Why Trace? Why Test?
- How Does Byteman Help
- Driving Byteman
- Questions
Byte(code) manipulation

- Available right now in a JVM near you
  - transform code/class structure at load
  - retransform code only after load
- java.lang.instrument
  - a pure byte(code)-bashing API
Byte(code) man(ipulation)

• Byteman makes it easy
  – inject Java code direct into Java code
    • directly refer to app/runtime data & types
    • dynamic linking, type-checking/inference

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

• Byteman makes it reversible
  – no structural changes
Example Byteman Rule

- Rule based scripting language
  - simple structure for injected code
  - very Java oriented (almost all Java)

```plaintext
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
```
Event Condition Action Rules

- Event
  - CLASS/INTERFACE, METHOD, AT
    - defines *trigger point(s)* for rule

- Binding
  - introduces/initializes rule variables

- Condition
  - Java boolean expression

- Action
  - one or more Java expressions
Example Byteman Rule 2

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

• inject through interface
• inject down into overriding methods
  – Executor, AbstractExecutor, ThreadPoolExecutor
• THROW/RETURN from trigger method
  – must conform to method contract
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
  – Byteman Rule Language
• Driving Byteman
• Questions
Location Clauses

AT ENTRY
AT EXIT
AT/AFTER READ [[package.]type.]field [count]
AT/AFTER WRITE [[package.]type.]field [count]
AT/AFTER READ $localvar [count]
AT/AFTER WRITE $localvar [count]
AT/AFTER CALL [[package.]type.]method
                   [(Types)] [count]
AT THROW [count]
AT LINE number
public boolean check(Sym sym)
  throws BadSym, BadType
{
  String s = "";              // AFTER WRITE $s
  if (badSym(sym)) {
    s = munge(sym.name);
    throw new BadSym(s);
  } else if (badType(sym.type)) {
    s = munge(sym.type.name);
    throw new BadType(s);
  }
  ...

Location Match Examples

```java
public boolean check(Sym sym) throws BadSym, BadType {
    String s = "";              // AFTER WRITE $s
    if (badSym(sym)) {
        s = munge(sym.name);      // AT WRITE $s 2
        throw new BadSym(s);
    } else if (badType(sym.type)) {
        s = munge(sym.type.name); // AFTER WRITE $s 3
        throw new BadType(s);
    }
    ...
```
Location Match Examples

```java
public boolean check(Sym sym)
    throws BadSym, BadType
{
    String s = "";
    if (badSym(sym)) {  // AT READ name 1
        s = munge(sym.name);
        throw new BadSym(s);
    } else if (badType(sym.type)) {  // AT READ name 2
        s = munge(sym.type.name);  // AT READ Type.name 1
        throw new BadType(s);
    }
    ...
```
public boolean check(Sym sym)
    throws BadSym, BadType
{
    String s = "";
    if (badSym(sym)) {
        s = munge(sym.name);      // AT CALL munge
        throw new BadSym(s);
    } else if (badType(sym.type)) {
        s = munge(sym.type.name); // AT CALL munge 1
        throw new BadType(s);
    }
    . . .
public boolean check(Sym sym)
  throws BadSym, BadType
{
  String s = "";
  if (badSym(sym)) {
    s = munge(sym.name);
    // AT THROW
    throw new BadSym(s); // AT THROW ALL
  } else if (badType(sym.type)) {
    s = munge(sym.type.name);
    // AT THROW 2
    throw new BadType(s); // AT THROW ALL
  }
  ...
Location Match Examples

```java
public boolean check(Sym sym)
    throws BadSym, BadType {
    String s = "";
    . . .
    if (sym.isBuiltIn()) {
        return true; // AT EXIT
    }
    . . .
    return false; // AT EXIT
}
```
Expressions

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

• Special variables
  – $*, $# trigger method param array/count
  – $! stacked return val AT EXIT/AFTER CALL
  – $@ stacked args AT CALL
  – $^ stacked throwable AT THROW
Expressions

• Full set of Java operations
  – operators + - / *, |, ||, ==<>, new, =, etc
  – instance/static field access & method calls
    • n.b. protected/private are ignored
  – built-in methods
  – no control structures

• $1 = "Andrew"$ updates trigger method
  – $loopvar = $loopvar + 1$
  – $! = 3$
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
  – Byteman Built-in Methods
• Driving Byteman
• Questions
Built-in Method Families

- **Tracing**
  - traceOpen, traceClose, traceIn, traceStack, ...

- **Managing Shared Rule State**
  - flag, clear, countDown, incrementCounter, ...

- **Timing**
  - createTimer, getElapsedTime, resetTimer, ...
Built-in Method Families

• Checking Caller Stack
  – callerEquals, callerMatches

• Thread Synchronization
  – delay, waitFor, signalWake, rendezvous, ...

• Recursive Trigger Management
  – setTriggering
Example Byteman Rule 3

```java
@MessageDriven(activationConfig = 
   {@ActivationConfigProperty(propertyName = 
      "cronTrigger", propertyValue = "0/5 * * * * ?")})
@ResourceAdapter("quartz-ra.rar")
public class AnnotatedQuartzMDBBean implements Job {

   private static final Logger log = 
       Logger.getLogger(AnnotatedQuartzMDBBean.class);
   public void 
       execute(JobExecutionContext jobExecutionContext) 
           throws JobExecutionException 
           {
               AnnotatedQuartzMDBBean.log.info("*************
* here in annotated!!!");
           }
}
```
Example Byteman Rule 3

RULE set up rendezvous
CLASS AnnotatedQuartzMDBBean
METHOD <init>
AT EXIT
IF NOT isRendezvous("quartz", 3)
DO createRendezvous("quartz", 3, true)
ENDRULE

RULE rendezvous 3 threads
CLASS AnnotatedQuartzMDBBean
METHOD execute
IF isRendezvous("quartz", 3)
DO traceln("**** enter quartz rendezvous");
   traceln("**** exit " + rendezvous("quartz"));
ENDRULE
Example 3
Demo
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
  – Rule Helpers
• Driving Byteman
• Questions
Helper Classes

• Built-ins are POJO public methods
  – take a look
    • org.jboss.byteman.rule.Helper

• You can use any POJO you like

```java
class DBHelper {
    public void trace(String msg, Record rec);
    ...
```
Helper Classes

RULE use my trace built-in
CLASS org.my.db.Manager
METHOD update(Record)
AT CALL setName(String)
HELPER org.my.bmutil.DBHelper
IF $@[1] == "Andrew"
DO trace("interesting!", $1)
ENDRULE
Helper Classes

• Often better to extend default helper
  – lets you to also use standard built-ins
    class DBHelper extends Helper

RULE use my trace built-in
CLASS org.my.db.Manager
METHOD update(Record)
AT CALL setName(String)
HELPERS org.my.bmutil, DBHelper
IF flagged($1) && $@[1] == "Andrew"
DO trace("interesting!", $1)
ENDRULE
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
• Driving Byteman
• Questions
Driving Byteman

• From the java command line
  - `javaagent:/path/to/agent.jar=agentoptions`
  - main byteman jar is an agent jar
    `$\{BYTEMAN_HOME\}/lib/byteman.jar`
  - various options available
    `=script:myrules.btm,listener:true`
• see manual for full details
Driving Byteman

- `bmjava [-l script.btm] org.my.DBApp`
- `bminstall pid|mainClass`
- `bmsubmit myrules.btm`
- `bmsubmit -u [myrules.btm]`
- `bmsubmit`
- `bmcheck -cp my.jar myrules.btm`
Driving Byteman

• From a Java program
  – `org.jboss.byteman.agent.install.Install`
    • main and install
  – `org.jboss.byteman.agent.submit.Submit`
    • main, addRulesFromFile and addScripts

• Used by contrib packages
  • dtest instrument remote JVM for test validation
  • BMUnit integrate Byteman into JUnit/TestNG
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
  – BMUnit package
• Questions
BMUnit Contrib Package

• Integrates Byteman into JUnit/TestNG
  – automatically loads agent into test JVM
  – automatically loads and unloads rules

• Annotate test classes or @Test methods
  – @BMScript identifies script file
  – @BMRule defines rule in annotation body
  – class level – load/unload around all tests
  – method level – load/unload around one test
BMUnit Contrib Package

• JUnit integration via @RunWith
  @RunWith(BMUnitRunner.class)
  class DBTests {
  ...

• TestNG integration via extends
  class DBTests extends BMNGRunner {
  ...
package org.my.dbtests;
@RunWith(BMUnitRunner.class)
@BMScript(value="traceRules",dir="scripts")
class DBTest1 {
    @Test
    @BMRule(className="FileInputStream",
            methodName="\n init\n (File)\n",
            condition="$1.getName().contains("Andy")",
            action="THROW new FileNotFoundException()"")
    public void testDBFileHandler() {
        ...
    }
}
BMUnit From ant Or maven

• Execution just needs jars in 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

• Declare as maven test dependencies
  – find them in the JBoss repo (1.5.1+)
  – surefire needs AdditionalClasspathElement
    • ${java.home}/../lib/tools.jar
Agenda

• Why Trace? Why Test?
• How Does Byteman Help
• Driving Byteman
• Questions
Questions

• Project Page
  – http://www.jboss.org/byteman
  – downloads
  – documentation
  – user & developer forums
  – code repository