Byteman

Using Bytecode Manipulation to Automate Multi- Threaded Testing

Andrew Dinn
JBoss, a Division of Red Hat

7460

JAZOON09
THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY
JUNE 22–25, 2009 ZURICH

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Testing Multi-Threaded Applications

- 'Application' means post-component integration
  - reliable, repeatable automation is hard to achieve
  - hard to rig test scenarios
    - code goes its own way
  - developing and maintaining test code is labour intensive
  - testing often 'moves the goal posts'
- Threads add Timing Problems
- synchronization is hard to get right
- test runs may *always* fail to display timing issues
  - test runs may *sometimes* fail to display timing issues
  - testing *always* 'moves the goal posts'
    - where and how far?

Testing Multi- Threaded Applications continued

- Test case: JBoss Web Services Transactions (XTS) recovery
  - client and web service threads (possibly distributed)
  - transaction service threads (possibly distributed)
    - message handler and message reply handlers
    - asynchronous service implementation threads
    - message resends
- Byteman was developed to help automate test runs
  - based on 'fault injection'
    - tests release code with *no* rewriting/ stubbing or recompilation
    - minimally invasive
  - employs script language based on Java
    - familiar, easy to use, powerful and flexible

Fault Injection

- Introduce variety of side effects into an application
  - Inject faults
    - break a specific part of the application in a known way
    - e.g. crash JVM on entry to `phase2Commit()`
    - throw `SystemException` from 2nd call to `prepare()`
  - Manage fault propagation
    - monitor and maintain conditions defined in the test scenario
    - e.g. suspend caller of `aborted()` until `ROLLBACK` resent twice
  - Trace execution
    - validate progress and outcome of test
    - e.g. log TX status at return from `phase2Commit()`
- May do code transformation offline or online
- May also require runtime support to execute side effects

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Byteman

- JBoss Bytecode Manipulation project
  - Byteman employs a Java agent to rewrite bytecode at load time
  - see java.lang.instrument package for details
- Side effects are defined offline in scripts
  - agent reads scripts during bootstrap and transforms any matching code
    - may extend to allow runtime (re)transformation
- Scripts comprise a sequence of Event Condition Action rules
  - simple structured way of defining where to introduce side effects
  - quick and easy to write and execute
  - flexible enough to configure complex test scenarios
  - script language based on Java
  - includes library of 'built- in' operations
    - extensible/ redefinable

ECA Rules

Event: *when* to run the side effects

- when control reaches a 'trigger' location
- just means some identifiable point in the application code
- n.b. Byteman events also 'bind' data derived from the trigger context

Condition: *whether* to run the side effects

- just a Java expression (including 'built-in' calls)
- bindings allows condition to be highly specific

Action: *what* side effects should be run

- just a sequence of Java expressions (including 'built-in' calls)
- possibly ending with a `return` or `throw`
  - i.e. rules can also alter control flow of trigger method
  - must conform to method signature

Simple Test Program

```
public class Test
{
  private int value = 0;
  private String name;
  public Test(String name) { this.name = name; }
  public int getValue() { return value; }
  public String getName() { return name; }
  // should be synchronized!
  public void increment(int threadId)
  {
    int newValue = value + 1;
    value = newValue
  }
  . . .
```

Simple Byteman Script

```
# simple Byteman script
RULE create rendezvous
CLASS Test
METHOD <init>
AT RETURN
BIND test : Test = $0,
     name : String = test.getName()
IF name.equals(“THREADSAFE?”)
DO debug("creating rendezvous for " + name),
   createRendezvous(test, 2)
ENDRULE
```

Simple Byteman Script Continued

```
# simple Byteman script
RULE rendezvous before write
CLASS Test
METHOD increment(int)
AT WRITE value
BIND test : Test = $0,
     id = $1
IF isRendezvous(test, 2) &&
   debug(“thread “ + id + “ rendezvous for “ + test.getName())
DO rendezvous(test),
   debug(“newValue = “ + $newValue)
ENDRULE
```

Simple Test Program continued

```
  . . .
  public static void main(String[] args) {
    final Test theTest = new Test("THREADSAFE?");
    Thread thread1 = new Thread() {
      public void run() { theTest.increment(1); }
    };
    Thread thread2 = new Thread() {
      public void run() { theTest.increment(2); }
    };
    thread1.start(); thread2.start();
    try { thread1.join(); thread2.join(); }
      catch (InterruptedException e) { /*ignore*/ }
    System.out.println("value is " + theTest.getValue());
  }
}
```

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Simple Test Demo

```
javac -g Test.java
```

```
java Test
   value is 2
```

usually prints value 2

```
java -javaagent:byteman.jar=script:TestScript.txt \
       -Dorg.jboss.byteman.debug Test
   rule.debug{create rendezvous} : creating rendezvous for
   THREADSAFE?rule.debug{rendezvous before write} : thread 1 rendezvous for
   THREADSAFE?
   rule.debug{rendezvous before write} : thread 2 rendezvous for THREADSAFE?
   rule.debug{rendezvous before write} : newValue = 1
   rule.debug{rendezvous before write} : newValue = 1
   value is 1
```

always prints value 1

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Event Locations

Location: identifies point in trigger method

```
AT ENTRY/RETURN
AT/AFTER READ value
AT/AFTER WRITE Account.total
AT/AFTER CALL com.acme.Foo.length()
AT LINE 103
```

Optionally supply a count for `READ`, `WRITE` and `CALL`

```
AT READ com.acme.Account.total 3
```

n.b. count refers to *lexical* not runtime order

Event Locations continued

Where specified package/ class/ method names, signatures etc are matched

Where absent they are inferred by inspecting the candidate class

e.g.

```
CLASS Foo
   METHOD test
   AT CALL length() 3
```

matches `org.acme.Foo.test()` and `org.my.Foo.test(int)`

matches 3rd call in `test` to any of `*.length()`

```
   String.length()
   org.acme.Foo.length()
```

partial location matches are ignored silently

Successful location match drives expression type inference and checking

expression type inference/ check failures are notified

Expressions in Bindings, Conditions & Actions

- Bound variable references
  - this, $0, and method parameters, $1, $2, etc
  - local vars in scope at the trigger location $i, $newValue etc
  - variables introduced in BIND, name, id, etc
- The usual Java operations are supported
  - static field references and static or instance method calls
  - all the normal operators, &&, ||, !, ^, |, &, +. -, /, *, %, ?:, [], etc
    - *except* new and = are (currently) disallowed
- Built-in operations
  - a standard suite of helper methods
  - default helper mostly targeted at thread management
  - easily extended or redefined

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Built- In Methods

```
# Thread management

void waitFor(Object id)
void waitFor(Object id, int millisecs)

boolean waiting(Object id)

boolean signalWake(Object id)
boolean signalThrow(Object id)

boolean signalWake(Object id, boolean mustMeet)
boolean signalThrow(Object id, boolean mustMeet)

boolean delay(int millisecs)
```

Built- In Methods continued

```
# Thread management continued

boolean createRendezVous(Object id, int expected)
boolean createRendezVous(Object id, int expected,
                         boolean restartable)

boolean isRendezVous(Object id, int expected)
int getRendezVous(Object id, int expected)

int rendezvous(Object id)

boolean killJVM()
```

Built- In Methods continued

```
# State management

boolean addCountDown(Object id, int count)
boolean countDown(Object id)
boolean isCountDown(Object id)

boolean flag(Object id)
boolean flagged(Object id)
boolean clear(Object id)
```

Built- In Methods continued

```
# State management continued

boolean createCounter(Object id)
boolean createCounter(Object id, int initial)

int readCounter(Object id)
int incrementCounter(Object id)
int decrementCounter(Object id)

boolean deleteCounter(Object id)
```

Built- In Methods continued

```
# Trace and debug

boolean openTrace(Object id, String filename)
boolean openTrace(Object id)
boolean closeTrace(Object id)

# “out” and “err” are always open and cannot be closed

boolean trace(Object id, String message)
boolean traceln(Object id, String message)

boolean debug(String message)
```

AGENDA

Testing Multi-Threaded Applications

Byteman

Simple Example Demo

Byteman Language

Byteman Built-Ins

Customizing Byteman

Questions

Helper Classes

Built-in methods are defined by public API of a POJO

- default is class `Helper`
- built-ins map 1-1 to instance methods of this class

Helper may be redefined per rule

- allows definition of test-specific conditions/actions
- keeps rules simple and clear
- insert `HELPER <classname>` before location specifier
- type check calls against your class
- engine calls your code during rule execution

Often useful to *extend* `Helper`

- allows standard built-ins to be *supplemented*
- or, if you don't like the default behaviour, *redefined* or *specialised*

Helper Classes continued

- Helper class is instantiated when rule is triggered
  - actually a generated subclass implementing `HelperAdapter`
- `HelperAdapter` provides interface to rule engine
  - allows bindings to be installed
    - that's why you need an instance *per-triggering*
  - generated methods include `execute()` method
    - *either* interprets rule parse tree
    - *or* calls generated bytecode (`execute0()`)
- Built-in calls are redirected to instance method calls
  - instance can access rule object and bindings (via `HelperAdapter`)
  - instance can retain and manage state across calls/triggerings
    - e.g. Waiters, Rendezvous, Flags, Counters etc

Summary

Testing Multi-Threaded Applications can benefit from tooling

Byteman is a clear, easy-to-use and powerful test scripting tool

- simple, declarative rules
- independent of application code
- sensitive to runtime context

Byteman aids resolution of timing issues

- introduce determinacy
- simulate real-world delays
- repeatable testing

Byteman language is easily extensible and redefinable

- test application-specific validation
- maintain simple, minimal rules

Andrew Dinn
http://www.jboss.org/byteman/

JBoss, A Division of Red Hat adinn@redhat.com

Questions