ORACLE

Programmer's Guide to the JDK Flight Recorder

Joakim Nordström
Sustaining Engineer
Java Product Group, Oracle
November 2023

JDK Flight Recorder

- Records events from the JVM, the JDK and applications
  - unified way to inspect the entire software stack
  - correlate events across different software layers and components
- Part of the JVM
- Low overhead
- Designed to be ”always on”

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

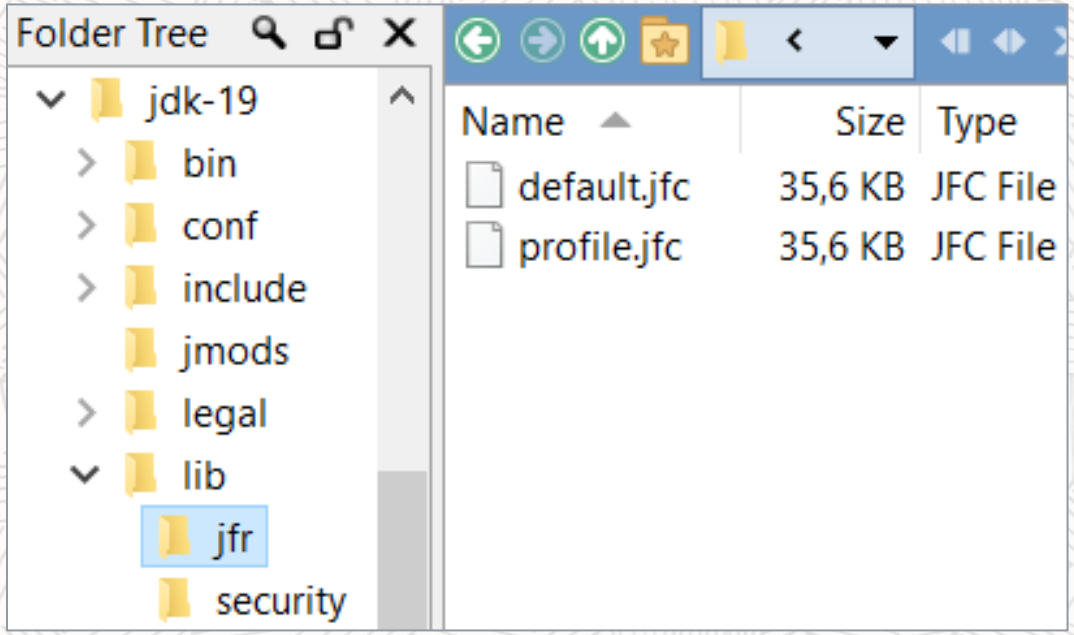
Consuming events

5

Using events in unit-test

Designed to be always on

- Two profiles shipped with the JVM
  - default.jfc
    - less than 1% overhead
    - designed for continous recording, ”always on”
  - profile.jfc
    - can have slightly more overhead (<2%)
    - slightly changed thresholds
    - a few more stack traces



Tip: don’t alter these files!

- Make copies if needed
  - Use `jfr` commmandline tool!
- Store custom settings together with app settings

Designed to be always on

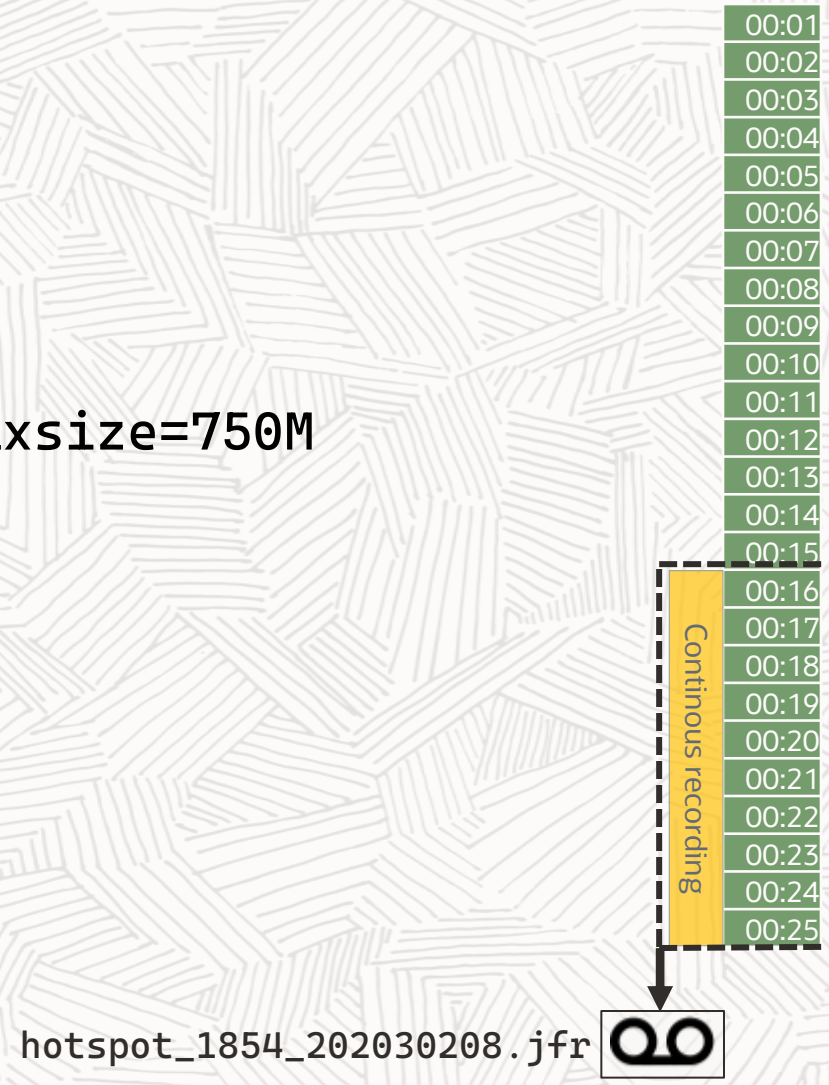
- Always start your JVM with Flight Recording!
  - Default profile has less than 1 % overhead

```
java -XX:StartFlightRecording
```

Designed to be always on

- Always start your JVM with Flight Recording!
  - Default profile has less than 1 % overhead

```
java -XX:StartFlightRecording:maxage=10m,maxsize=750M
```

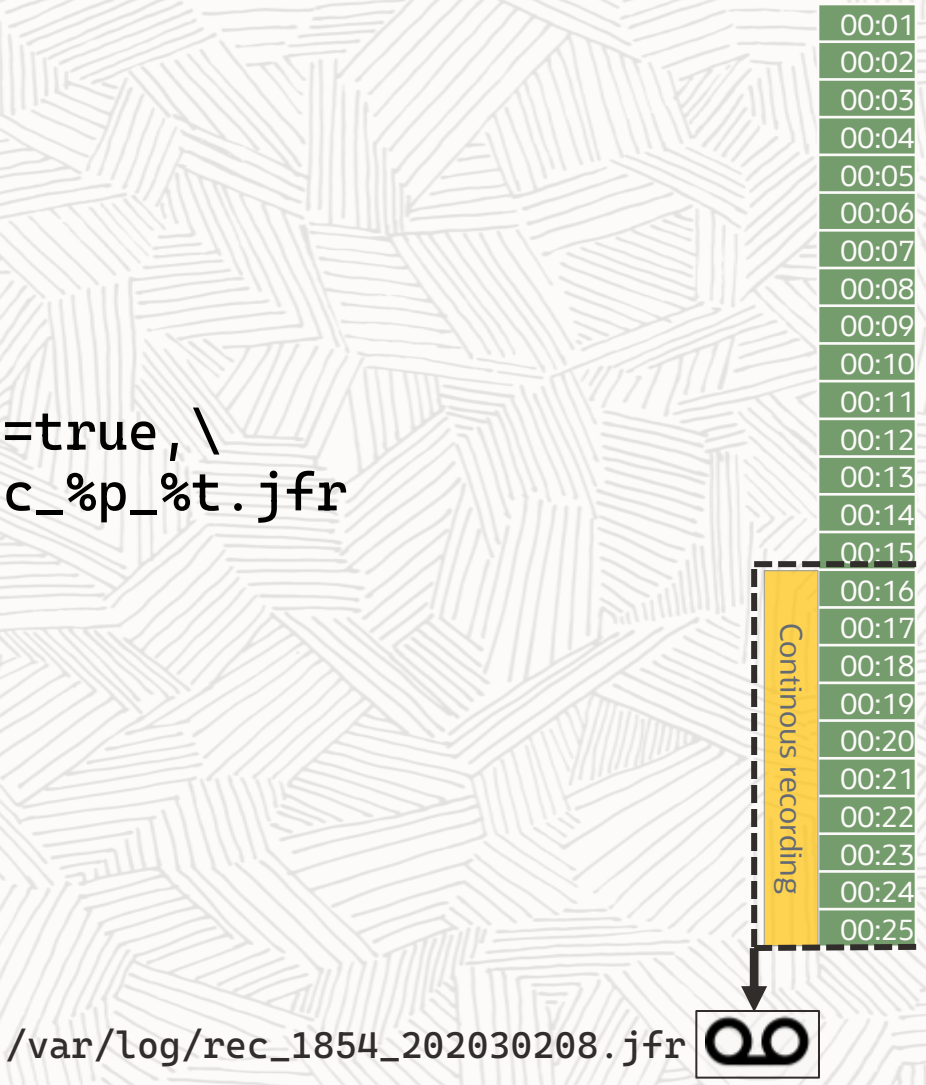


Designed to be always on

- Always start your JVM with Flight Recording!
  - Default profile has less than 1 % overhead

```
java -XX:StartFlightRecording:dumponexit=true,\
                    filename=/var/log/rec_%p_%t.jfr
```

- **`dumponexit=true`**
  - will write JFR on JVM exit
- default name: **`hotspot_<pid>_<timestamp>.jfr`**
  - **`filename`** can be
    - directory
    - filename
    - **`%p`** for Process ID
    - **`%t`** for timestamp



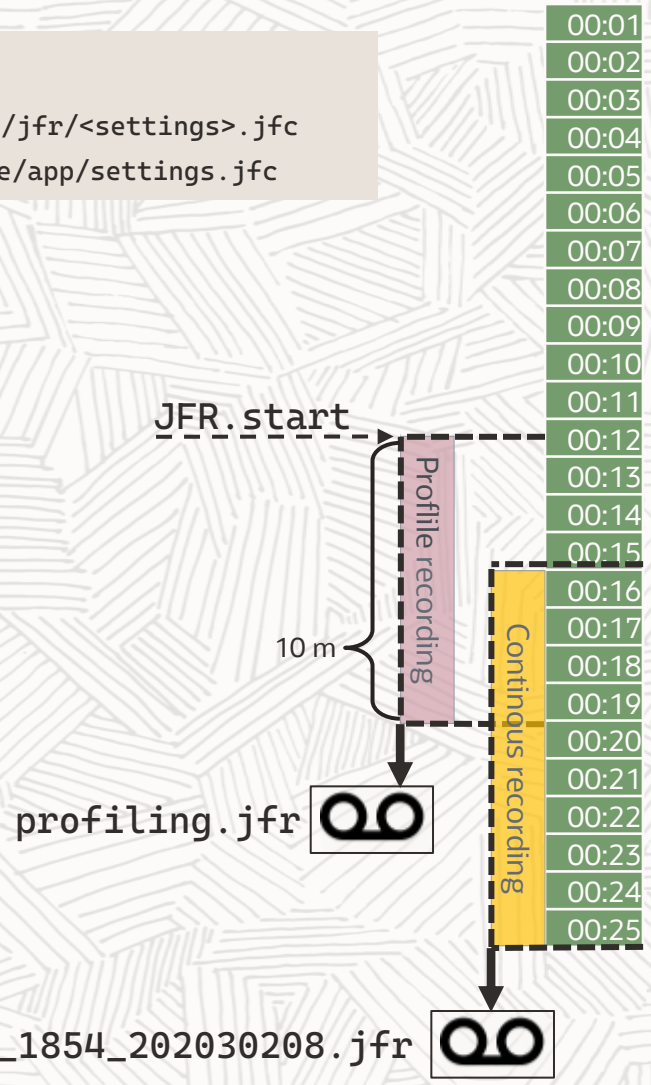
Start a recording during runtime

- Start recording using the “profile” setting:

```
$ jcmd <pid> JFR.start settings=profile \
      filename=profiling.jfr \
      duration=10m
```

settings can be

- file in `/$JAVA_HOME/lib/jfr/<settings>.jfc`
- arbitrary file, f.i. `/home/app/settings.jfc`



Using JCMD to control JFR recordings

- Start recording

```
$ jcmd <pid> JFR.start
```

- Write recording to disk

```
$ jcmd <pid> JFR.dump
```

- Stop recording

```
$ jcmd <pid> JFR.stop
```

- Check active recordings

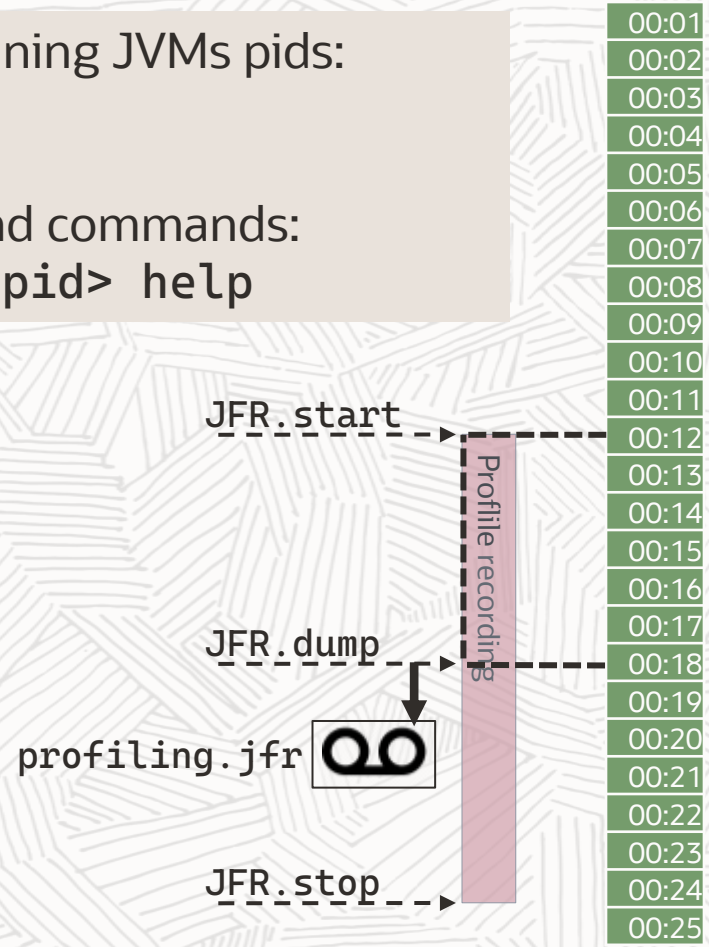
```
$ jcmd <pid> JFR.check
30496:
Recording 1: name=1 maxsize=250,0MB (running)
```

List all running JVMs pids:

```
$ jcmd
```

List all jcmd commands:

```
$ jcmd <pid> help
```



Agenda

1 Start/stop recordings

2 Analyzing recordings

3 Create and use custom events

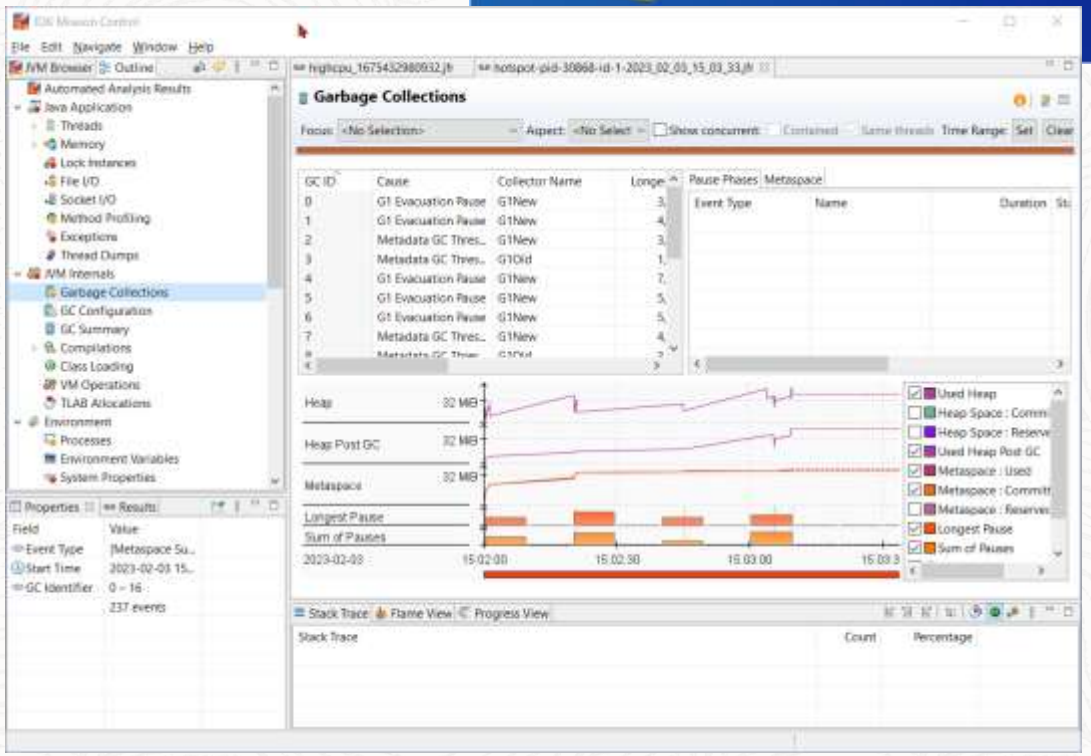
4 Consuming events

5 Using events in unit-test

Examining the JFR recording

JDK Mission Control 8, “JMC”

- Graphical interface to analyze recordings
- Separate download: oracle.com/java/technologies/jdk-mission-control.html



Examining the JFR recording

JFR commandline tool

- Available in the JDK
- List events, output as text/JSON
- Create .jfc configuration files
- Disassemble chunks from recording
  - If you have very large JFR recording
- Assemble leftover JFR chunks
- Scrub events from recording

Programmer's guide to JDK Flight Recorder

```
JSNORDST@JSNORDST-P53 /cygdrive/C/Users/JSNORDST/dev/
$ jfr summary hotspot-pid-34312-id-2-2023_01_31_09_41_54.jfr

 Version: 2.1
 Chunks: 7
 Start: 2023-01-31 08:38:46 (UTC)
 Duration: 188 s

 Event Type                                    Count  Size (bytes)
======================================================================
 jdk.ActiveSetting                              6580        215517
 jdk.NativeMethodSample                         5981         71772
 jdk.JavaMonitorWait                            5857        152294
 jdk.ModuleExport                               5306         57281
 jdk.BooleanFlag                                3675        112287
 jdk.SystemProcess                              2552        126355
 jdk.LongFlag                                   1386         46039
 jdk.ModuleRequire                              1071         10710
 jdk.NativeLibrary                              1022         61600
 jdk.UnsignedLongFlag                           1022         34909
 jdk.InitialEnvironmentVariable                  518         36897
```

Viewing on-going JFR recordings

JDK 21

JFR "views" shows aggregated event data

- On-going, live recordings

  `jcmd <pid> JFR.view <view>`

- File recordings

  `jfr view <view> recording.jfr`

JDK 21 has 70 predefined views

Inside the Java Dumpster (InsideDumpster)

```
JSNORDST@JSNORDST-P53 /cygdrive/d/jsnordst/Bugs/InsideDumpster
$ jfr view --verbose gc-pauses server_diagnosis_2023_10_18_11_34_AM.jfr

GC Pauses
---------

Total Pause Time: 41,3 ms

Number of Pauses: 10

Minimum Pause Time: 0,200 ms

Median Pause Time: 3,97 ms

Average Pause Time: 4,13 ms

P90 Pause Time: 6,87 ms

P95 Pause Time: 6,76 ms

P99 Pause Time: 6,76 ms

P99.9% Pause Time: 6,76 ms

Maximum Pause Time: 6,76 ms
```

Read more: https://egahlin.github.io/2023/05/30/views.html

```
$ jfr help view
Java virtual machine views:
 class-modifications        gc-concurrent-phases longest-compilations
 compiler-configuration     gc-configuration     native-memory-committed
 compiler-phases            gc-cpu-time          native-memory-reserved
 compiler-statistics        gc-pause-phases      safepoints
 deoptimizations-by-reason  gc-pauses            tlabs
 deoptimizations-by-site    gc-references        vm-operations
 gc                         heap-configuration

Environment views:
 active-recordings         cpu-information       jvm-flags
 active-settings           cpu-load              native-libraries
 container-configuration   cpu-load-samples      network-utilization
 container-cpu-throttling  cpu-tsc               recording
 container-cpu-usage       environment-variables system-information
 container-io-usage        events-by-count       system-processes
 container-memory-usage    events-by-name        system-properties

Application views:
 allocation-by-class    exception-count       native-methods
 allocation-by-site     file-reads-by-path    object-statistics
 allocation-by-thread   file-writes-by-path   pinned-threads
 class-loaders          finalizers            socket-reads-by-host
```

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

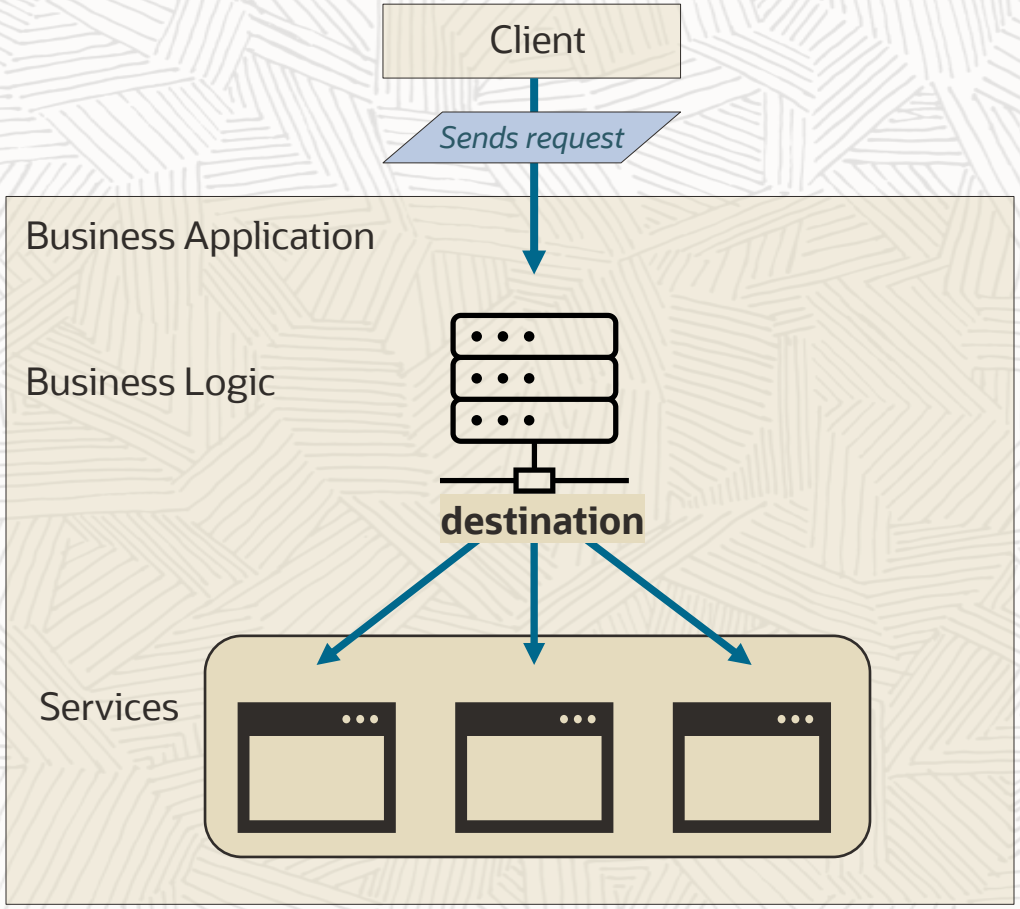
Our Business Application -- Inside the Java Dumpster

Mimics a real world system

- Server application
  - (Jetty, Micronaut, Tomcat, WebLogic, etc)
- JDK 21

Client sends request to Business Application

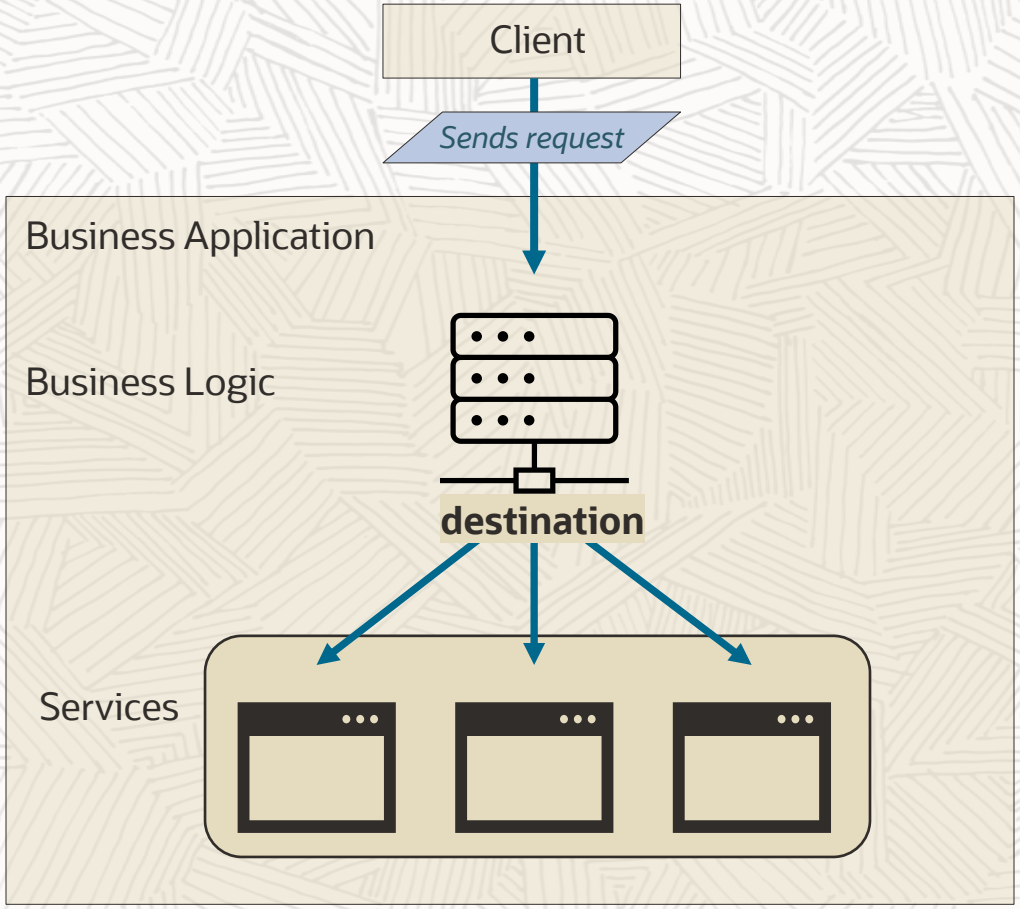
Business Logic decides based on `destination` which `service` should handle the request



Our Business Application -- Inside the Java Dumpster

Requirements

- Monitor service calls
- Catch unhandled destinations



```java
package inside.dumpster.monitoring.event;



public class ServiceCallEvent {

  public String destination;



  public Class serviceImplClass;
}
```

```java
package inside.dumpster.monitoring.event;



public class ServiceCallEvent extends jdk.jfr.Event {

  public String destination;



  public Class serviceImplClass;
}
```

```java
package inside.dumpster.monitoring.event;

@Name("inside.dumpster.ServiceCall")
@Label("Service Call")
@Category({"Business Application", "Services"})
public class ServiceCallEvent extends jdk.jfr.Event {
  @Label("Service Destination")
  public String destination;

  @Name("serviceClass")
  @Label("Service Implementation Class")
  public Class serviceImplClass;
}
```

`@Name`
- unique
- use reverse domain-name notation
  - `com.acme.Application`

`@Label`
- capitalized
- human-readable

`@Name` (for a field)
- start with lowerCase

`@Category`
- capitalized
- descriptive
- unique
- create a logic hierarchy

```java
public BusinessLogicService lookupService(Destination destination) {
  final BusinessLogicService service;




  switch(destination) {
      case Comp3:
          service = new UploadImageService();
          break;
      ...
  }



  return service;
}
```

```java
public BusinessLogicService lookupService(Destination destination) {
  final BusinessLogicService service;

  ServiceCallEvent serviceCallEvent = new ServiceCallEvent();
  serviceCallEvent.destination = destination.name();
  serviceCallEvent.begin();
  switch(destination) {
      case Comp3:
          service = new UploadImageService();
          break;
      ...
  }
  serviceCallEvent.serviceImplClass = service.getClass();
  serviceCallEvent.commit();

  return service;
}
```

```java
public BusinessLogicService lookupService(Destination destination) {
  final BusinessLogicService service;

  ServiceCallEvent serviceCallEvent = new ServiceCallEvent();
  serviceCallEvent.destination = destination.name();
  serviceCallEvent.begin();
  switch(destination) {
      case Comp3:
          service = new UploadImageService();
          break;
      ...
      case Unknown:
      default:
          UnhandledServiceCallEvent unhandledServiceEvent = new UnhandledServiceCallEvent();
          unhandledServiceEvent.destination = destination.name();
          unhandledServiceEvent.commit();

          return null;
  }
  serviceCallEvent.serviceImplClass = service.getClass();
  serviceCallEvent.commit();

  return service;
}
```

```java
public BusinessLogicService lookupService(Destination destination) {
  final BusinessLogicService service;

  ServiceCallEvent serviceCallEvent = new ServiceCallEvent();
  serviceCallEvent.destination = destination.name();
  serviceCallEvent.begin();
  switch(destination) {
      case Comp3:
          service = new UploadImageService();
          break;
      ...
      case Unknown:
      default:
          UnhandledServiceCallEvent unhandledServiceEvent = new UnhandledServiceCallEvent();
          unhandledServiceEvent.destination = destination.name();
          unhandledServiceEvent.commit();

          return null;
  }
  serviceCallEvent.serviceImplClass = service.getClass();
  serviceCallEvent.commit();

  return service;
}
```

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

JDK Mission Control

File Edit Navigate Window Help

JVM Browser | Outline

- Automated Analysis Results
- Java Application
  - Threads
  - Memory
  - Lock Instances
  - File I/O
  - Socket I/O
  - Method Profiling
  - Exceptions
  - Thread Dumps
- JVM Internals
  - Garbage Collections
  - GC Configuration
  - Compilations
  - Class Loading
  - VM Operations
  - TLAB Allocations
- Environment
  - Processes
  - Environment Variables
  - System Properties
  - Native Libraries
  - Recording
- Event Browser

diagnosis_2022_10_11_10_02_AM.jfr

Event Browser

<No Selection> Aspect: <No Selection> Show concurrent: Contained Same threads

Event Types Tree

Search the tree

- Business Application 525
- Flight Recorder 1 725
- Java Applicatio
- Java Developm
- Java Virtual Ma
- Operating Syste

Start Time	Event Type
2022-10-11 10:02:36	Object Allocation Sample
2022-10-11 10:03:00	Object Allocation Sample
2022-10-11 10:03:05	Object Allocation Sample

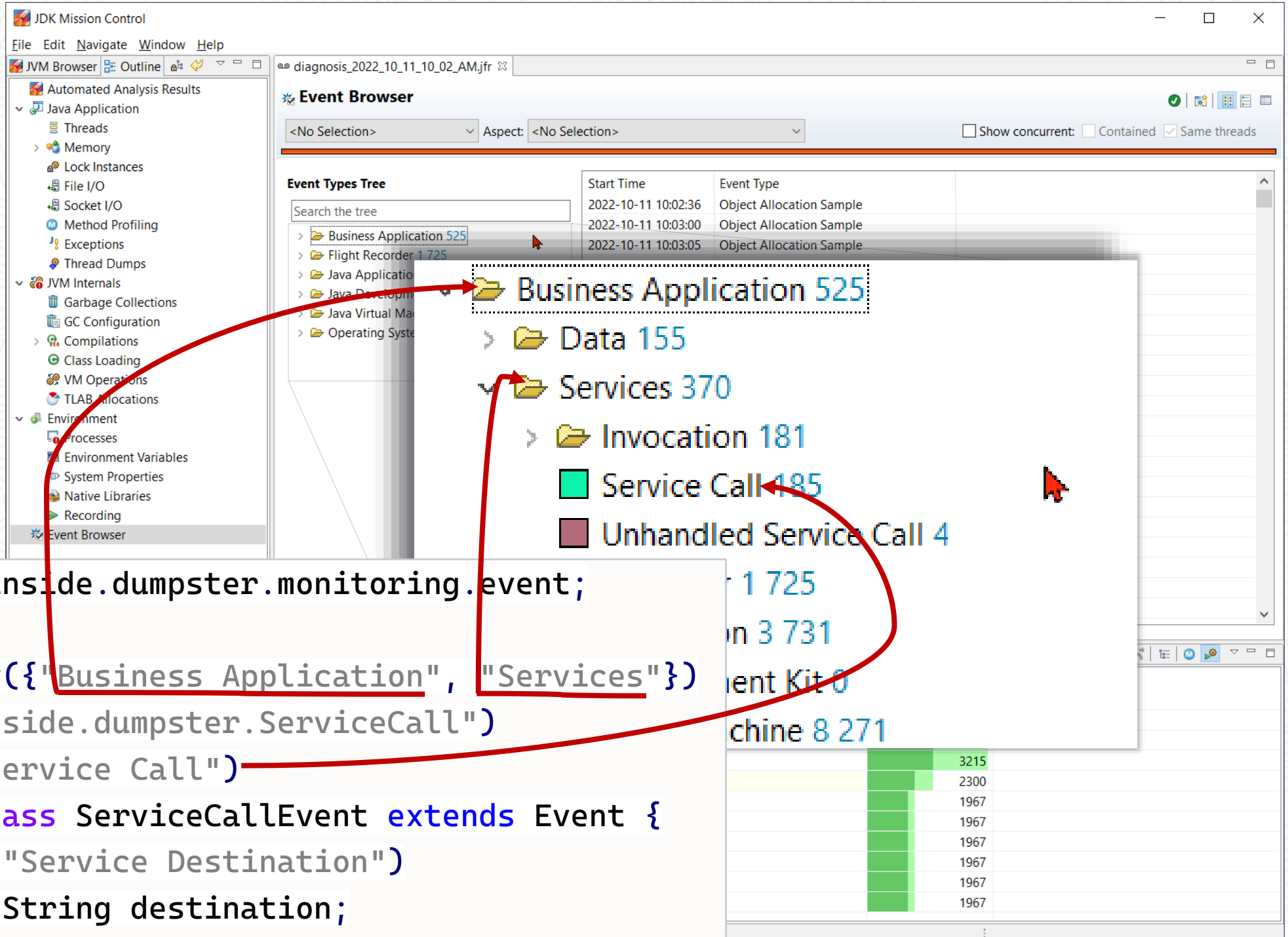
- Business Application 525
  - Data 155
  - Services 370
    - Invocation 181
    - Service Call 185
    - Unhandled Service Call 4
  - Flight Recorder 1 725
  - Java Application 3 731
  - Java Development Kit 0
  - Java Virtual Machine 8 271

Properties | Results

0	Biased Locking Revocation
N/A	DMS Incidents

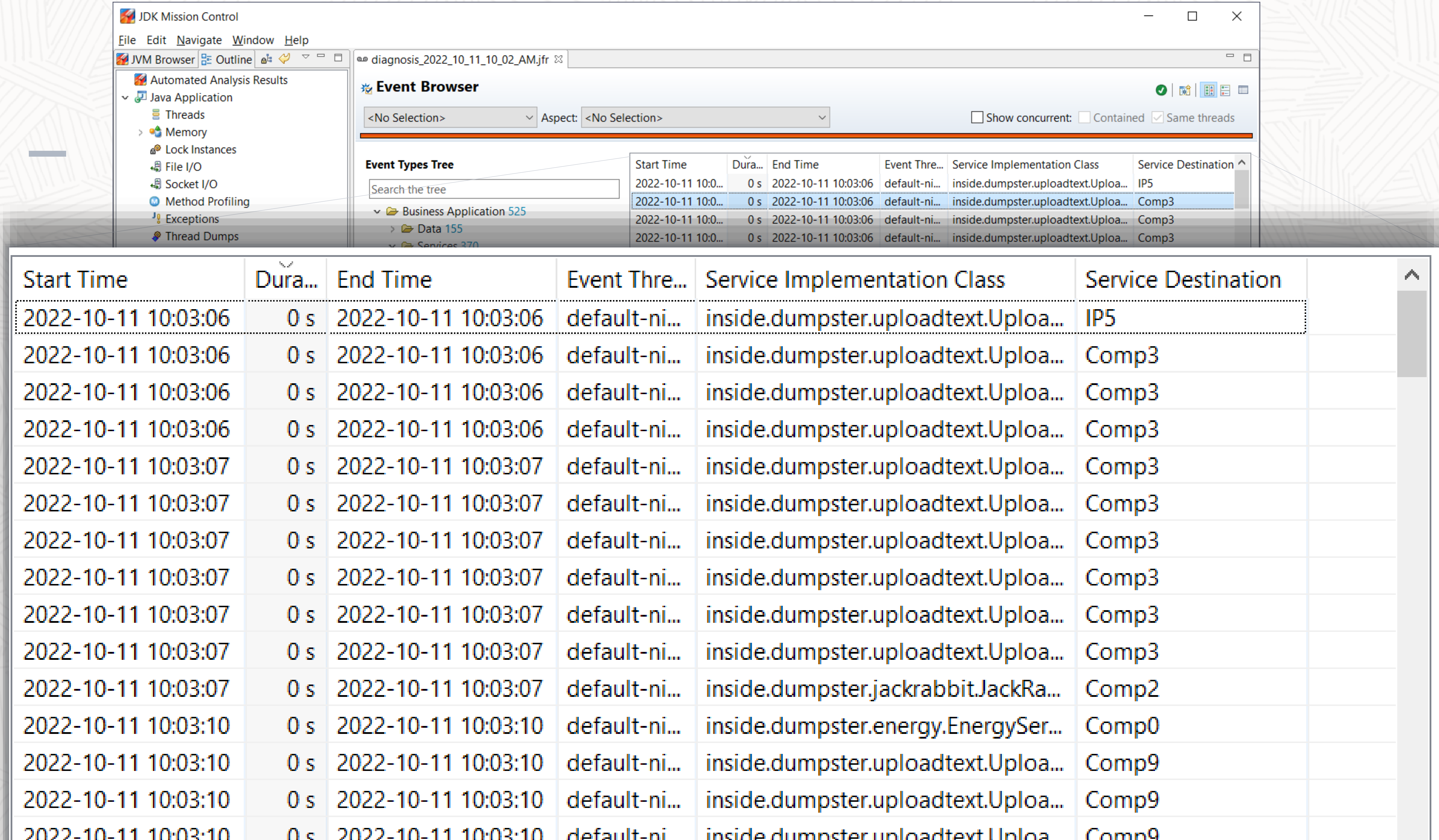
Stack Trace

Stack Trace	
void java.lang.Thread	
void io.netty.util.conc	
void io.netty.util.inter	
void io.netty.util.concurrent.SingleThreadEventExecutor$4.run():986	3215
void io.netty.channel.nio.NioEventLoop.run():460	2300
int io.netty.channel.nio.NioEventLoop.select(long):813	1967
int io.netty.channel.nio.SelectedSelectionKeySetSelector.select():68	1967
int sun.nio.ch.SelectorImpl.select():146	1967
int sun.nio.ch.SelectorImpl.lockAndDoSelect(Consumer, long):129	1967
int sun.nio.ch.WEPollSelectorImpl.doSelect(Consumer, long):111	1967
int sun.nio.ch.WEPoll.wait(long, long, int, int):-1	1967




```
package inside.dumpster.monitoring.event;

@Category({"Business Application", "Services"})
@Name("inside.dumpster.ServiceCall")
@Label("Service Call")
public class ServiceCallEvent extends Event {
  @Label("Service Destination")
  public String destination;
```



Start Time	Dura...	End Time	Event Thre...	Service Implementation Class	Service Destination
2022-10-11 10:0...	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	IP5
2022-10-11 10:0...	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:0...	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:0...	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3

Start Time	Dura...	End Time	Event Thre...	Service Implementation Class	Service Destination
2022-10-11 10:03:06	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	IP5
2022-10-11 10:03:06	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:06	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:06	0 s	2022-10-11 10:03:06	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp3
2022-10-11 10:03:07	0 s	2022-10-11 10:03:07	default-ni...	inside.dumpster.jackrabbit.JackRa...	Comp2
2022-10-11 10:03:10	0 s	2022-10-11 10:03:10	default-ni...	inside.dumpster.energy.EnergySer...	Comp0
2022-10-11 10:03:10	0 s	2022-10-11 10:03:10	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp9
2022-10-11 10:03:10	0 s	2022-10-11 10:03:10	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp9
2022-10-11 10:03:10	0 s	2022-10-11 10:03:10	default-ni...	inside.dumpster.uploadtext.Uploa...	Comp9

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

Consuming events programmatically

`jdk.jfr.consumer.EventStream`

- Stream events from recording

`jdk.jfr.Recording`

- Configure, start, stop, dump recording

`jdk.jfr.consumer.RecordingStream`

- Stream events

`jdk.management.jfr.RemoteRecordingStream`

- Stream events from remote source

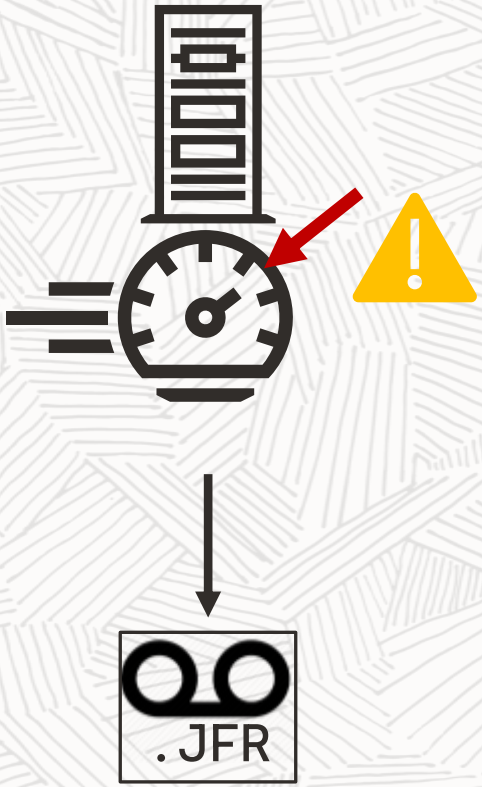
Monitoring CPU level using RecordingStream

Problem

- Our JVM takes up a lot of CPU, and we don't know why

Action

- When the CPU load is high, dump a JFR recording to analyze why this happens



Monitoring CPU level using RecordingStream

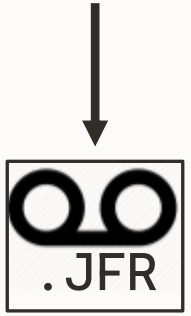
```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.     Configuration conf = Configuration.getConfiguration("default");
8.     try (var stream = new RecordingStream( conf )) {
9.
10.        stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.        stream.onEvent("jdk.CPULoad", event -> {
12.           float cpuLoad = event.getFloat("jvmUser");
13.           if (cpuLoad > 0.8) {
14.              stream.dump(dumpPath);
15.           }
16.        });
17.        stream.start();
18.    }
19.  }
```

RecordingStream has no events enabled without Configuration

- enable events explicitly, or
- add Configuration

Since we want a JFR file to analyze, we want as many events as possible – use Configuration

Monitoring CPU level using RecordingStream



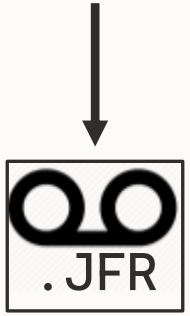
Inside the Java Dumpster (InsideDumpster)

```
$ jfr view thread-cpu-load highcpu_1675432980932.jfr

                                Thread CPU Load

Thread                                                            System  User
----------------------------------------------------------------- ------ -----
CreditCrunch 17                                                    0,00% 3,34%
CreditCrunch 16                                                    0,00% 3,15%
CreditCrunch 11                                                    0,00% 3,13%
CreditCrunch 4                                                     0,00% 3,09%
CreditCrunch 5                                                     0,00% 3,08%
CreditCrunch 10                                                    0,00% 2,98%
CreditCrunch 0                                                     0,00% 2,82%
N/A                                                                0,00% 2,65%
CreditCrunch 0                                                     0,00% 2,58%
CreditCrunch 1                                                     0,00% 2,56%
CreditCrunch 2                                                     0,00% 2,49%
CreditCrunch 9                                                     0,00% 2,48%
CreditCrunch 14                                                    0,00% 2,45%
CreditCrunch 6                                                     0,00% 2,40%
CreditCrunch 15                                                    0,00% 2,35%
CreditCrunch 12                                                    0,00% 2,28%
CreditCrunch 2                                                     0,00% 2,26%
CreditCrunch 6                                                     0,00% 2,20%
```

Monitoring CPU level using RecordingStream



JDK Mission Control

File Edit Navigate Window Help

JVM Browser | Outline

- Automated Analysis Results
- Java Application
  - Threads
    - (Legacy) Threads
  - Memory
  - Lock Instances
  - File I/O
  - Socket I/O
  - Method Profiling
  - Exceptions
  - Thread Dumps
- JVM Internals
  - Garbage Collections
  - GC Configuration

Properties | Results

Field	Value
	0 events

highcpu_1675432980932.jfr

Automated Analysis Results

Java Application

36 High JVM CPU Load

Lock Instances

27 Context Switches

The program causes many context switches during the recording.
The program context switched a lot and many threads waited on the same monitor. Consider using fewer threads, or decreasing lock contention by other means.

68 Java Blocking

Threads in the application were blocked on locks for a total of 50,488 s.
The most blocking monitor class was "java.io.PrintStream", which was blocked 51 times for a total of 47,673 s.The following regular expression was used to exclude threads from this rule: "(.*weblogic\.socket\.Muxer.*)"

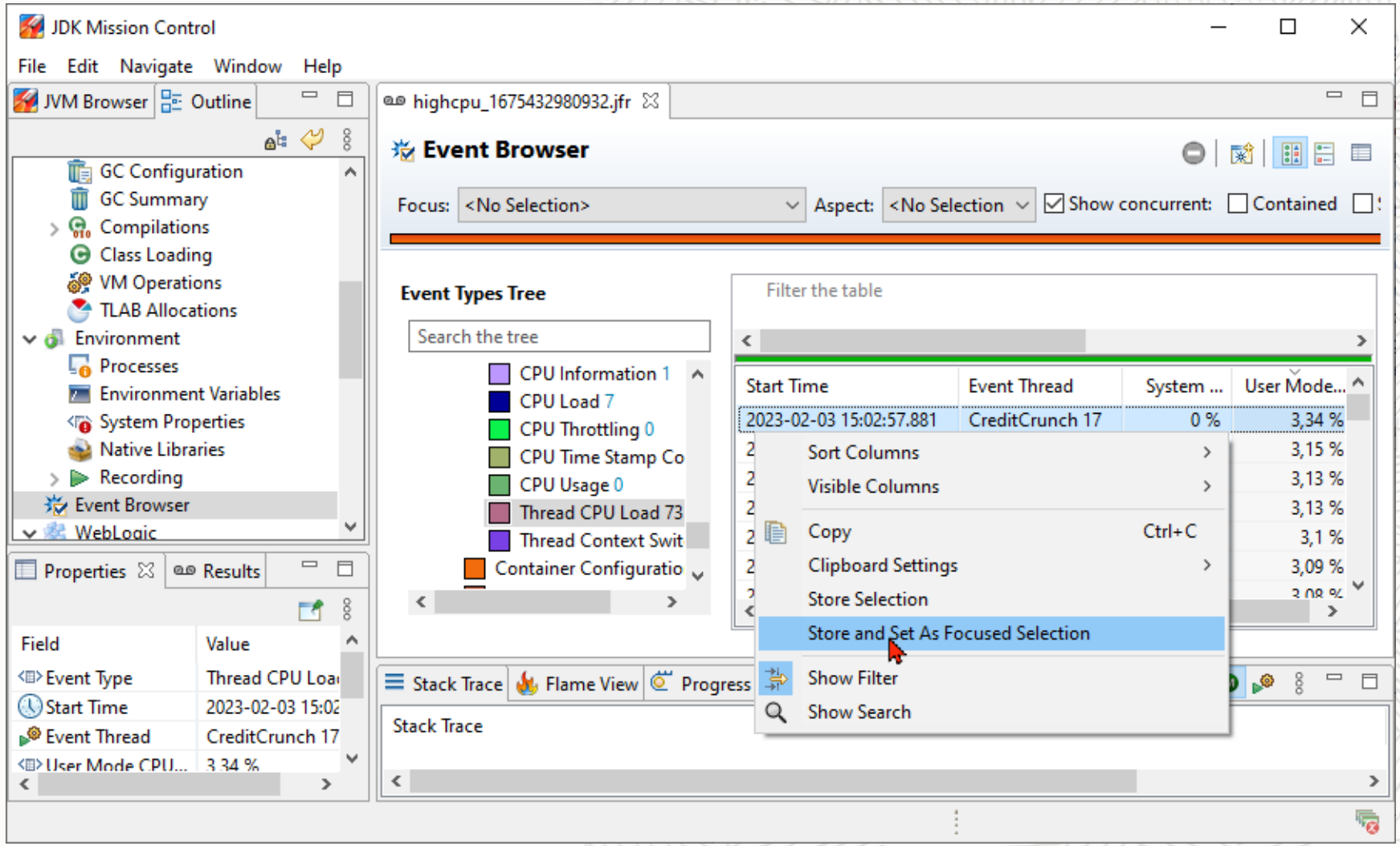
JVM Internals

25 Discouraged Management Agent Settings

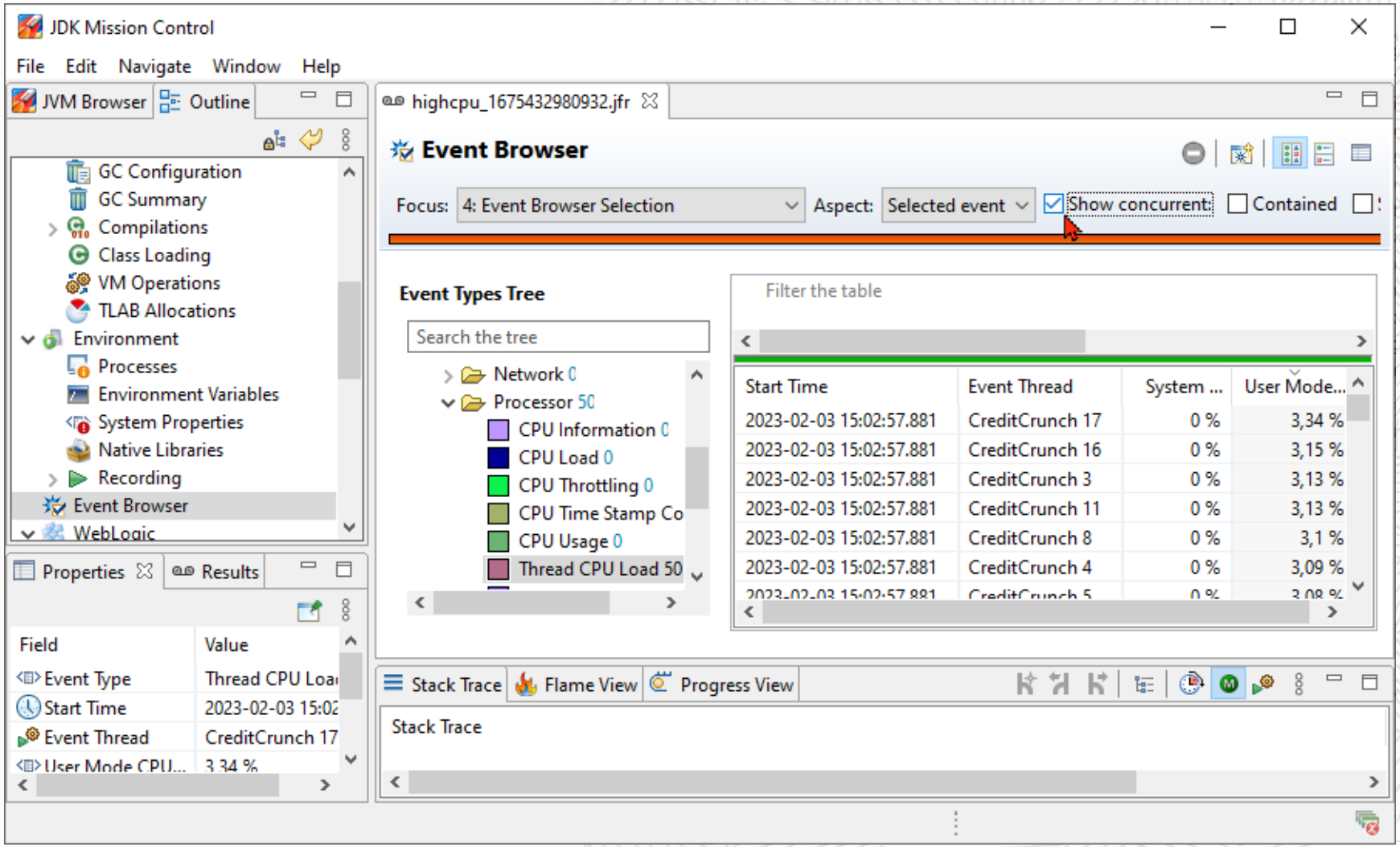
Stack Trace | Flame View | Progress View

Loading events from final.jfr (Finished at 15:06)
Could not load events from D:\final.jfr

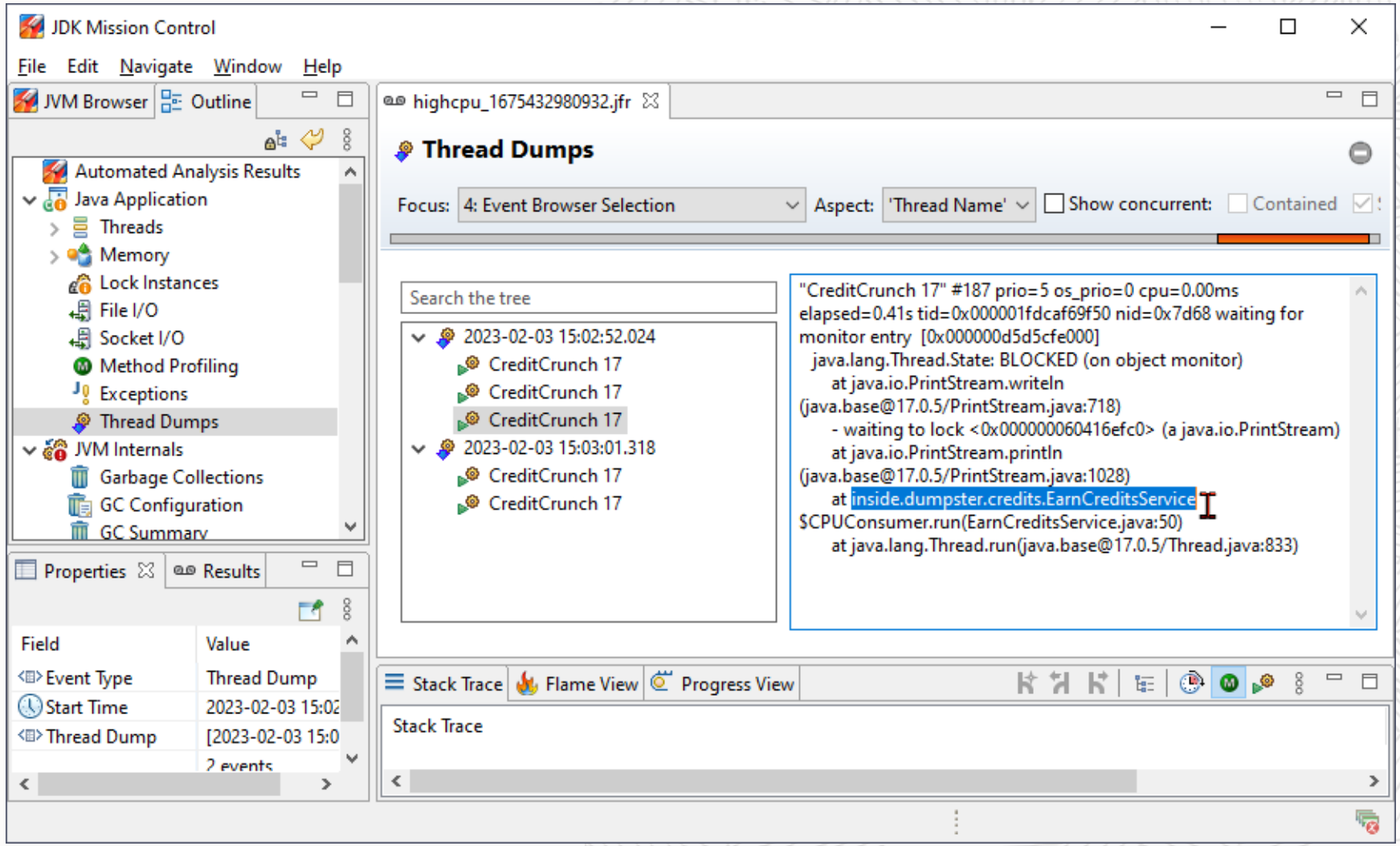
Monitoring CPU level using RecordingStream



Monitoring CPU level using RecordingStream



Monitoring CPU level using RecordingStream



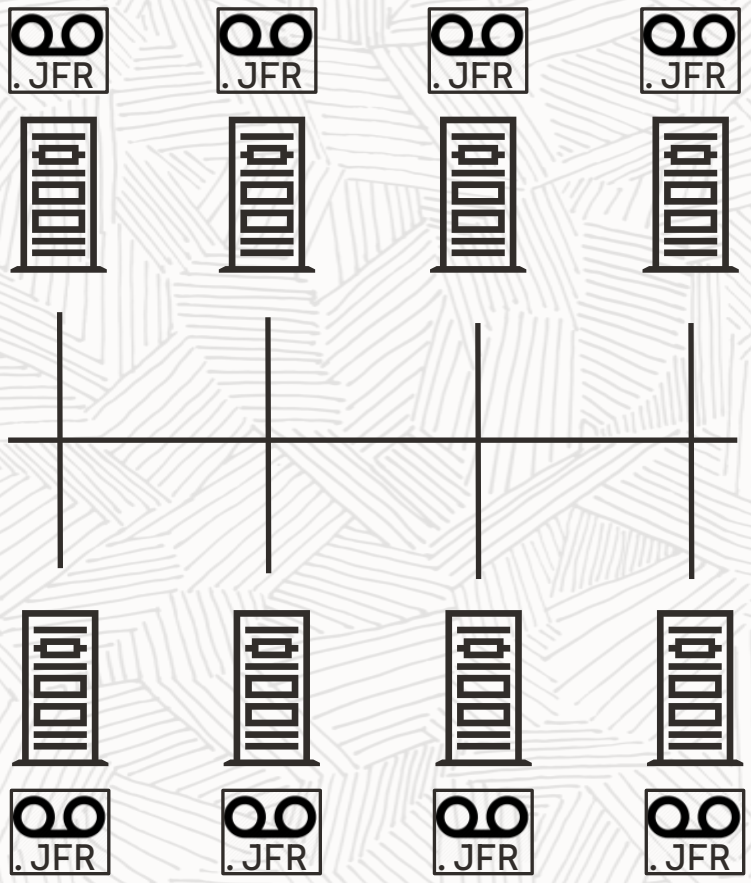
Remotely monitoring CPU level using RemoteRecordingStream

Problem

- We're saving JFR recordings on high CPU, but our server nodes are shortlived, and automatically removed

Solution

- Dump JFR recording on a more persistent node



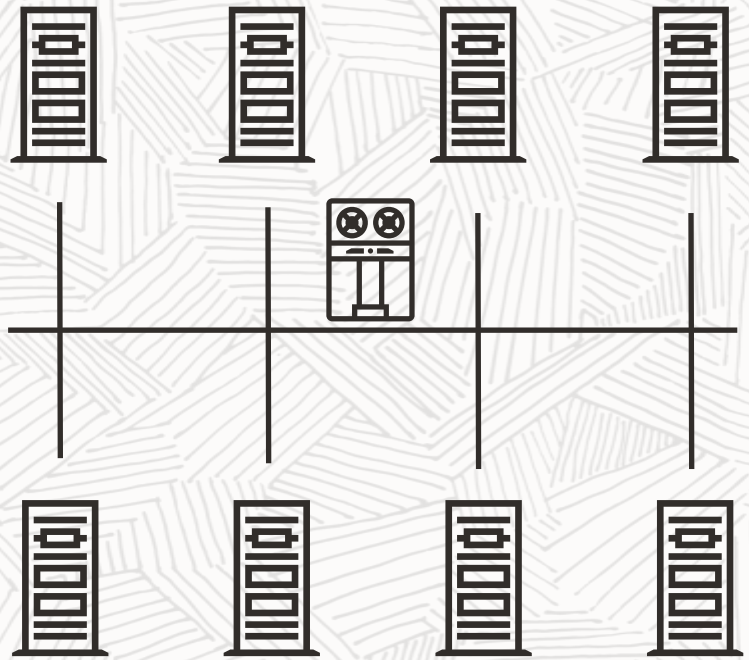
Remotely monitoring CPU level using RemoteRecordingStream

Problem

- We're saving JFR recordings on high CPU, but our server nodes are shortlived, and automatically removed

Solution

- Dump JFR recording on a more persistent node



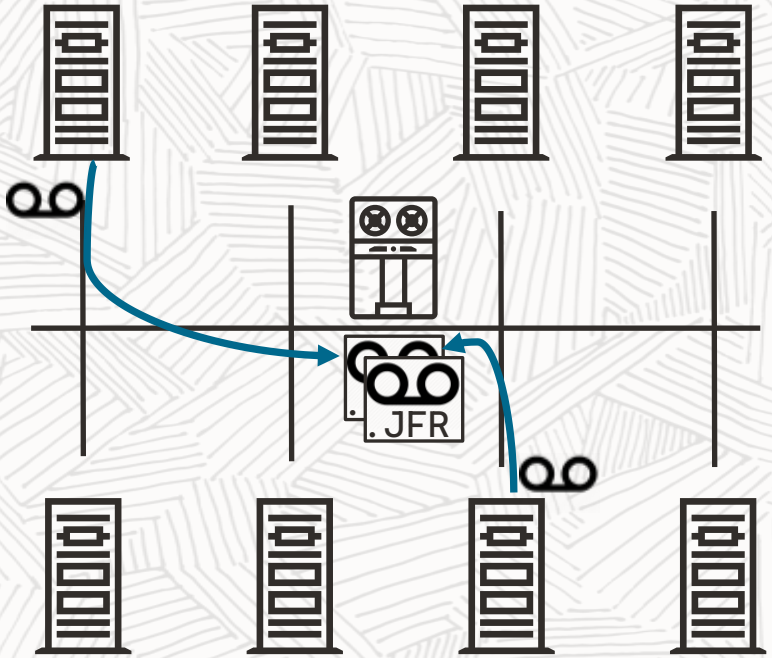
Remotely monitoring CPU level using RemoteRecordingStream

Problem

- We're saving JFR recordings on high CPU, but our server nodes are shortlived, and automatically removed

Solution

- Dump JFR recording on a more persistent node



Remotely monitoring CPU level using RemoteRecordingStream

```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.
8.     try (var stream = new RecordingStream()) {
9.
10.        stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.        stream.onEvent("jdk.CPULoad", event -> {
12.           float cpuLoad = event.getFloat(“jvmUser");
13.           if (cpuLoad > 0.8) {
14.              stream.dump(dumpPath);
15.           }
16.         });
17.         stream.start();
18.     }
19.   }
```

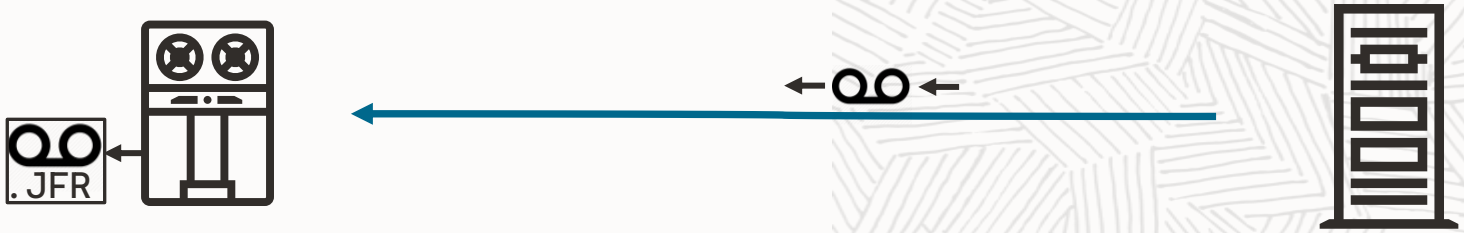
Remotely monitoring CPU level using RemoteRecordingStream

```
1.  void startCPULoadMonitor(String host, Path dumpPath) throws Exception {
2.      JMXServiceURL u = new JMXServiceURL(
3.              "service:jmx:rmi:///jndi/rmi://" + host + "/jmxrmi");
4.      JMXConnector c = JMXConnectorFactory.connect(u,
5.              Map.of("jmx.remote.credentials", new String[]{user, pwd}));
6.      MBeanServerConnection conn = c.getMBeanServerConnection();
7.
8.      try (var stream = new RemoteRecordingStream( conn )) {
9.          stream.setMaxAge(Duration.ofMinutes(10));
10.         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.         stream.start();
18.     }
19. }
```

Javadoc:
It's highly recommended that a max age or max size is set before starting the stream.

This means we'll keep 10 minutes worth of data on the client.

Remotely monitoring CPU level using RemoteRecordingStream



Client side

```
java -jar RemoteMonitor.jar server12:12345
```

Server side

```
java -XX:StartFlightRecording
 -Dcom.sun.management.jmxremote.port=12345
 -Dcom.sun.management.jmxremote.authenticate=true
 -Dcom.sun.management.jmxremote.password.file=jmx.pwd
 -Dcom.sun.management.jmxremote.access.file=jmx.acs
 -Dcom.sun.management.jmxremote.ssl=false
 -Djava.rmi.server.hostname=127.0.0.1
 -jar server.jar
```

JMX setup link:
https://docs.oracle.com/en/java/javase/21/management/monitoring-and-management-using-jmx-technology.html

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

Using JFR events in unit tests

JEP 421: Deprecate Finalization for Removal

Finalizers are deprecated for removal

New JFR event:

- `jdk.FinalizerStatistics` events are emitted for each instantiated class with a non-empty finalize() method

```
@Override
protected void finalize() throws Throwable {
  if (fw != null) fw.close();
  if (file != null) file.delete();
}
```

Problem

- Developers keep adding finalizers in their services.

Solution

- Add a testcase that captures all `jdk.FinalizerStatistics` events

Using JFR events in unit tests -- RecordingStream


```
1.  @Test
2.  public void testNoServicesUsesFinalizers throws Exception {
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.     assertTrue(classesWithFinalizers.isEmpty);
15. }
```

Using JFR events in unit tests -- RecordingStream


```
1.  @Test
2.  public void testNoServicesUsesFinalizers throws Exception {
3.     try (var stream = new RecordingStream()) {
4.       stream.enable("jdk.FinalizerStatistics");
5.       stream.onEvent("jdk.FinalizerStatistics", (event) -> {
6.         classesWithFinalizers.add(event.getClass("finalizableClass").getName());
7.       });
8.       stream.startAsync();
9.
10.      callAllServices();
11.
12.      stream.stop();        // new in JDK 20
13.    }
14.    assertTrue(classesWithFinalizers.isEmpty);
15. }
```

Using JFR events in unit tests -- Recording


```
1.  @Test
2.  public void testNoServicesUsesFinalizers throws Exception {
3.     try (Recording record = new Recording()) {
4.       record.enable("jdk.FinalizerStatistics");
5.       record.start();
6.       callAllServices();
7.       record.stop();
8.       record.dump(tempFile);
9.       try (EventStream stream = EventStream.openFile(tempFile)) {
10.        stream.onEvent("jdk.FinalizerStatistics", (event) -> {
11.          classesWithFinalizers.add(event.getClass("finalizableClass").getName());
12.        });
13.        stream.start();
14.      }
15.      Files.delete(tempFile);
16.    }
17.    assertTrue(classesWithFinalizers.isEmpty);
18. }
```

Using JFR events in unit tests -- JFRUnit

Not part of the JDK!

JFRUnit

- Framework by Gunnar Morling
- JDK 16 and up
- https://github.com/moditect/jfrunit

```
1.  public JfrEvents jfrEvents = new JfrEvents();
2.  @Test
3.  @EnableEvent(FinalizerStatistics.EVENT_NAME)
4.  public void testNoServicesUsesFinalizers() throws Exception {
5.
6.    callAllServices();
7.    jfrEvents.awaitEvents();
8.
9.    assertEquals(0, jfrEvents.filter(FINALIZER_STATISTICS).count());
10. }
```

Enabling/disabling events

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})


public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

Some other annotations:

```
@MemoryAddress
@Frequency
@Percentage
```

```
for (int pos : backend.getNext()) {

    byte arr[] = backend.read(pos);


    frontend.addBytes(arr);
}
```

Enabling/disabling events

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})


public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

```
for (int pos : backend.getNext()) {
    var event = new BytesProcessed();
    event.begin();
    byte arr[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();

    frontend.addBytes(arr);
}
```

Enabling/disabling events

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)

public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

```
for (int pos : backend.getNext()) {
    var event = new BytesProcessed();
    event.begin();
    byte arr[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();

    frontend.addBytes(arr);
}
```

Enabling/disabling events

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)

public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

  public String detail;
}
```

```
for (int pos : backend.getNext()) {
    var event = new BytesProcessed();
    event.begin();
    byte arr[] = backend.read(pos);
    event.end();
    event.bytes = b.length();

    if (event.isEnabled()) {
      // possibly costly call
      event.detail = backend.getDetails();
    }

    event.commit();

    frontend.addBytes(arr);
}
```

Enabling/disabling events

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)
@Threshold("50 ms")
public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

  public String detail;
}
```

```
for (int pos : backend.getNext()) {
    var event = new BytesProcessed();
    event.begin();
    byte arr[] = backend.read(pos);
    event.end();
    event.bytes = b.length();

    if (event.shouldCommit()) {
      // possibly costly call
      event.detail = backend.getDetails();
    }

    event.commit();

    frontend.addBytes(arr);
}
```

Enabling in a test

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)

public class BytesRead extends Event {
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

```
1.  @Test
2.  public void testBytesRead throws Exception {
3.     try (var stream = new RecordingStream()) {
4.        stream.enable("backend.io.BytesRead");
5.        stream.onEvent("backend.io.BytesRead",
6.           (event) -> {
7.              assertWhatever(event);
8.           });
9.        stream.startAsync();
10.       doStuff();
11.       stream.stop();
12.    }
13.    assertTrue(allIsGood);
14. }
```

Enabling for a recording

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)

public class BytesRead extends Eve
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

```
$ jcmd 23544 JFR.start                          \
       +backend.io.BytesRead#enabled=true
23544:
Started recording 6. No limit specified, using
maxsize=250MB as default.

Use jcmd 23544 JFR.dump name=6 filename=FILEPATH to
copy recording data to file.
```

Enabling in a configuration

```
@Name("backend.io.BytesRead")
@Label(”Bytes Read")
@Category({"Backend", ”IO"})
@Enabled(false)

public class BytesRead extends Eve
  @Label(”Bytes Read")
  @DataAmount
  public long bytes;

}
```

```
$ jfr configure                                \
      --input profile.jfc                      \
      +backend.io.BytesRead#enabled=true       \
      --output highprofile.jfc
Configuration written successfully to: highprofile.jfc

$ jcmd 23544 JFR.start settings=highprofile.jfc
23544:
Started recording 7. No limit specified, using
maxsize=250MB as default.
```

Agenda

1

Start/stop recordings

2

Analyzing recordings

3

Create and use custom events

4

Consuming events

5

Using events in unit-test

Questions?

Blog: jaokim.github.io
Github: github.com/jaokim/inside-java-dumpster
Twitter: @jaokim
Mastodon: @jaokim@techhub.social

eof