Harnessing Java Flight Recorder

Marcus Hirt
Consulting Member of Technical Staff

Agenda

- Flight Recorder Overview
- Producing Flight Recordings
- Analyzing Flight Recordings
  - Overview/Key indicators
  - Method Profiling
  - Memory Allocation

Agenda cont'd

  - GC analysis
  - Weblogic plug-in
  - Using the Operative Set
- Common pitfalls/misunderstandings
- Customization (unsupported)
  - Adding custom events (unsupported)
  - Customizing the GUI (unsupported)

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Flight Recorder

101

- High Performance Event Recorder
- Built into the JVM
- Binary recordings
- Chunks
  - Self contained
  - Self describing

Flight Recorder Helps You To...

- Resolve problems faster
- Find bottlenecks in your applications
- Find bottlenecks in ISV provided applications
  - Unhappy with the performance? Send your ISV a (JFR) recording.
- Do post mortem analysis, even from crash dumps

Flight Recorder Performance

Extremely Low Overhead

- Built into the JVM/JDK, by the people developing the JVM
- High performance flight recording engine and high performance data collection
  - Access to data already collected in the runtime
  - Thread local native buffers
  - Invariant TSC for time stamping
  - More accurate method profiling (method profiling data even from outside safe-points)
  - Faster and more accurate allocation profiling (scalarization not undone by profiler))

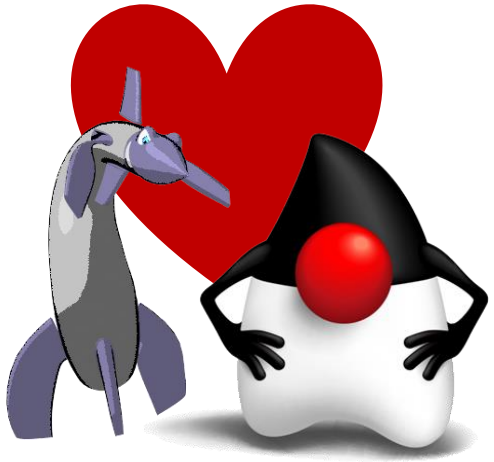
Flight Recorder History

- Started out with JRockit Runtime Analyzer
  - A means to get more information about the JVM and applications running on the JVM
- Addition to JRA – LAT
  - Latency Analyzer Tool
  - Soft real time GC (Deterministic GC) – important to discover non-GC related latencies
- Customer wanted always on capability – JRockit Flight Recorder
  - Low overhead imperative
  - Dump at any time to get data collected so far

Java Flight Recorder

JVM Convergence

- Already released the engine with 7u4
- Waited until 7u40 to release with JDK
  - Only a very limited number of Java and JVM events available in 7u4!
  - Lot's of WLDF (third party) events though



Different Kinds of Recordings

- Continuous Recordings
  - Have no end time
  - Must be explicitly dumped
- Time Fixed Recordings (sometimes known as profiling recordings)
  - Have a fixed time
  - Will be automatically downloaded by Mission Control when done (if initiated by Mission Control)

How to Think About Recordings

Recordings are collections of event type settings

- A “recording” can both mean an ongoing recording on the server side, as well as a recording file. The context usually separates the two.
- It might help to think of the server side recording as:
  - A named collection of event type settings…
  - …that are active for a certain period of time.

How to Think About Recordings

Example

- A continuous recording R_0 is started at T_0 with settings S_0. After a while, a time fixed recording R_1 is started at T_1 with settings S_1, where $S_1 \supset S_0$. The time fixed recording R_1 ends at T_2.
  This is what will be recorded:

Time	Settings
$T_0 \rightarrow T_1$	S_0
$T_1 \rightarrow T_2$	$S_0 \cup S_1$
$> T_2$	S_0

If dumping R_0 for a time range intersecting $[T_1,T_2]$, you will get information that you did not ask for in the settings (S_0). All this in the name of performance. Once T_2 arrives, the settings for R_1 will be popped, and we're back to just recording S_0.

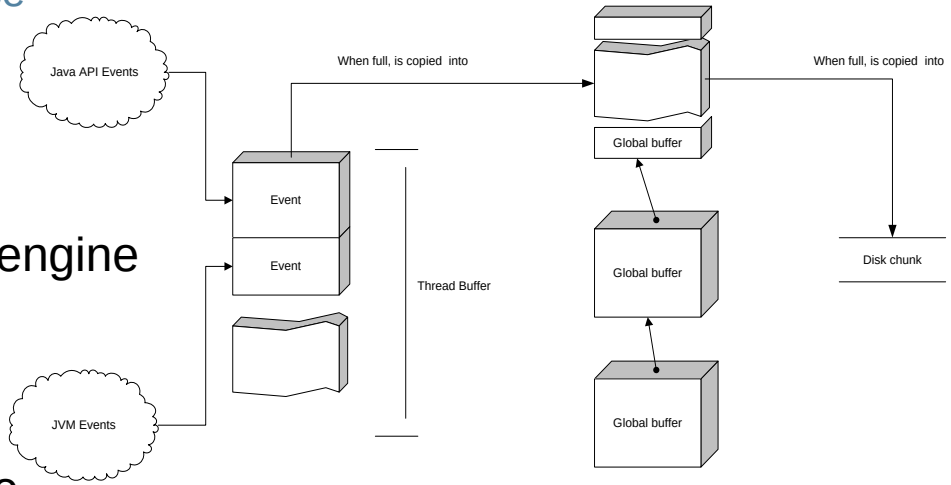
Different Kinds of Events

- Instant Event
  - Data associated with the time the data was captured
- Requestable Event
  - Polled from separate thread
  - Has a user configurable period
- Duration Event
  - Has a start time and a stop time
- Timed Event
  - Like Duration Event, but with user configurable threshold

Flight Recorder Inner Workings

Performance, performance, performance

- Extremely low overhead
  - Using data already gathered
  - High performance recording engine
- Testing!
- Third party events
  - WLS events already available
  - DMS events already available
  - Experimental JavaFX events
  - You can add your own! (Not supported yet.)



PRODUCING RECORDINGS

Preparations

Hotspot

- Need to have started the JVM from which to get recording with the appropriate flags (not required for 8u40 and later)

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder
```

- Need to have a recent enough JDK
  - 7u4, if having only WLS events are enough
  - 7u40 and later if Java & JVM events are of interest
- If remote monitoring is required:
  - Start with the appropriate com.sun.management flags
  - In 7u40 JDP can be used for easy discovery of manageable JVMs on the network

Creating Recordings Using Mission Control

Easy and intuitive

1. Find a JVM to do a recording on in the JVM Browser
2. Double click the Flight Recorder node under the JVM
3. Follow the wizard

(will show demo soon)

Note: Ongoing recordings are listed as nodes under the Flight Recorder node. To dump one, simply drag and drop the ongoing recording to the editor area, or double click it. This is mostly useful for continuous recordings.

Creating Recordings Using Startup Flags

Useful for enabling continuous recordings at startup

- Documentation of startup flags available in the JDK docs
- The following example starts up a 1 minute recording 20 seconds after starting the JVM:

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder -
XX:StartFlightRecording=delay=20s,duration=60s,name=MyRecording,filena
me=C:\TEMP\myrecording.jfr,settings=profile
```

  - The settings parameter takes either the path to, or the name of, a template
  - Default templates are located in the jre/lib/jfr folder.
  - ***Note:*** Using the settings parameter will require either a JRockit or a Hotspot **7u40 or later**.
- To get more information on what is going on, change the log level: `-XX:FlightRecorderOptions=loglevel=info`

The Default Recording

Shorthand for starting a continuous recording. With benefits.

- Special short hand to start the JVM with a continuous recording
- Started with `-XX:FlightRecorderOptions=defaultrecording=true`
- The default recording will have the recording id 0
- Only the default recording can be used with the dumponexit and dumponexitpath parameters
- The following example will start up the continuous recording. It will be dumped when the JVM exits to `C:\demos\dumponexit.jfr`.

```
 -XX:+UnlockCommercialFeatures -XX:+FlightRecorder -
XX:FlightRecorderOptions=defaultrecording=true,dumponexit=true,dumponexitpath=C:
\demos\dumponexit.jfr
```

Creating Recordings Using JCMD

Useful for controlling JFR from the command line

Usage: `jcmd <pid> <command>`

Example starting a recording:

```
jcmd 7060 JFR.start name=MyRecording settings=profile
  delay=20s duration=2m filename=c:\TEMP\myrecording.jfr
```

Example checking on recordings:

```
jcmd 7060 JFR.check
```

Example dumping a recording:

```
jcmd 7060 JFR.dump name=MyRecording
  filename=C:\TEMP\dump.jfr
```

RECORDING CREATION DEMO

ANALYZING RECORDINGS

Analyzing Flight Recordings in JMC

- All tab groups except for the general Events tab group are preconfigured to show a certain aspect of the recording (sometimes referred to as static or preconfigured tabs)
- The pre-configured tabs highlights various areas of common interest, such as code, memory & GC, threads and IO
- General Events tab group - useful for drilling down further and for rapidly homing in on a set of events with certain properties

The Operative Set

Power Feature

- The Operative Set is a global set of events
- Events can be added or removed to the operative set from the context menu
- The Events tabs usually have a check box to only show events in the operative set
- Using the Events tabs together with the Operative Set is a powerful way to home in on events with a certain set of properties

Common Features in Most Tabs

- Almost all tabs have a range selector at the top
  - Used to filter on a time range
  - Can be synchronized between all tabs
  - Highlights events in the operative set in cyan
- The tabs in the Events tab group can be used in conjunction with the Event Types view to filter on Event Types
- In filter boxes:
  - Kleene star can be used as wildcard (`*.sun.*`)
  - Start with regexp: if more power is needed (`regexp:.*\.sun\..*`) (not as performant though)

Programmatically Analyzing Flight Recordings

Using the JMC parser (Unsupported)

- The parser included in Mission Control can be used to analyze recordings programmatically
- The JMC team also have an experimental JDBC bridge
- More info on my blog: http://hirt.se/blog/?p=446

Some Common Pitfalls

- Not accounting for thresholds
  - Thresholds are very useful for keeping performance up but still detecting outliers
  - Can be confusing. Example:
    Thread T has been running for 2 minutes and sum of latencies is a minute. Was the thread T running unblocked for a minute?
- Not accounting for CPU load
  - Don’t make decisions based on method profiling data if there is no load
  - If you have full load, then looking at latencies may be a waste of time

ANALYZING RECORDINGS - DEMOS

CUSTOMIZATION

Adding Your Own Events (unsupported)

```
import com.oracle.jrockit.jfr.*;

public class Example {
    private final static String PRODUCER_URI = "http://www.example.com/demo/";
    private Producer    myProducer;
    private EventToken  myToken;

    public Example() throws URISyntaxException, InvalidEventDefinitionException, InvalidValueException {
        myProducer = new Producer("Demo Producer", "A demo event producer.", PRODUCER_URI);
        myToken = myProducer.addEvent(MyEvent.class);
    }

    @EventDefinition(path="demo/myevent", name = "My Event", description="An event triggered by doStuff.", stacktrace=true, thread=true)
    private class MyEvent extends TimedEvent {
        @ValueDefinition(name="Message", description="The logged important stuff.")
        private String text;
        public MyEvent(EventToken eventToken) {
            super(eventToken);
        }

        public void setText(String text) {
            this.text = text;
        }
    }

    public void doStuff() {
        MyEvent event = new MyEvent(myToken);
        event.begin();
        String importantResultInStuff = "";
        // Generate the string, then set it...
        event.setText(importantResultInStuff);
        event.end();
        event.commit();
    }
}
```

Minimizing Object Creation

- Can reuse event objects
- Use with care – only where you know it's thread safe

```
private MyEvent event = new MyEvent(myToken);
public void doStuffReuse() {
    event.reset();
    event.begin();
    String importantResultInStuff = "";
    // Generate the string, then set it...
    event.setText(importantResultInStuff);
    event.end();
    event.commit();
}
```

Built in GUI editor (unsupported)

- The JMC has a built in designer
- Can be used to both customize the existing GUI and produce entirely new GUIs for events
- The created GUIs can be exported as plug-ins and shared

CUSTOMIZATION DEMOS

FUTURE

JDK 9 / JMC 6

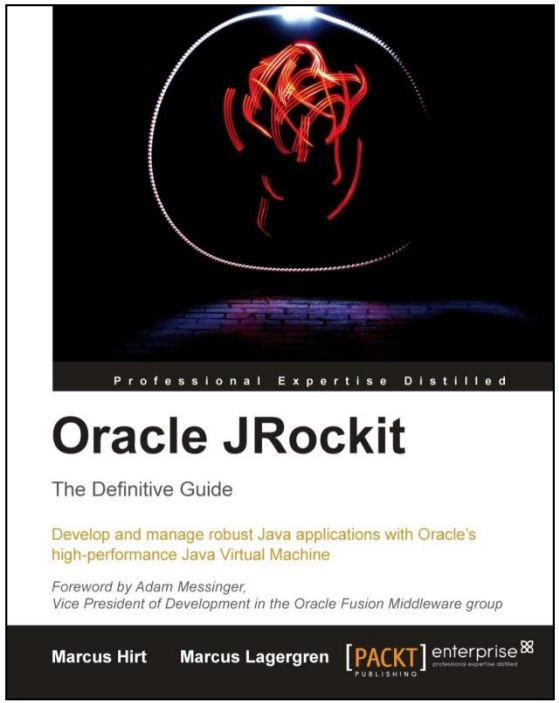
Released next year

- JDK 9
  - Supported POJO API for controlling the flight recorder (in process)
  - Supported JMX API for controlling the flight recorder (remotely)
  - Supported API for producing custom events
- JMC
  - Greatly revised user interface
  - Automated analysis of flight recordings

Other Resources

Plus shameless book plug

- JMC Homepage
  http://oracle.com/missioncontrol
- Hirt's blog
  http://hirt.se/blog
- Twitter
  @javamissionctrl, @hirt
- Shameless Book plug
- JMC Tutorial
  http://hirt.se/blog/?p=611



JavaOne ORACLE


JavaOne™