JMC/JFR: Under the hood

Profiling/Monitoring with joy

Miroslav Wengner

Safe Harbour Statement

All what you will hear can be different, this presentation is for motivational purposes …

Miroslav Wengner

- Husband, Father, Software Engineer, Technology Enthusiast
- OpenJDK Committer , Java Mission Control Project
- Co-Author of Robo4J Project (Duke Award)
- Contributor to other open-source projects
- Java Champion, JavaOne RockStar

OPEN VALUE

Application Crash

No Relevant Information

Adding a Logging

Performance Penalties

Removing Logging

Agenda

OPEN VALUE

- Brief history
- JFR in bullet points
- JFR fundamentals / Under the hood
- Performance (why overhead 1%)
- Power of Visualization (**DEMO**)
- Progress and Agent Plugin (**DEMO**)
- Q/A

Brief history : back in time

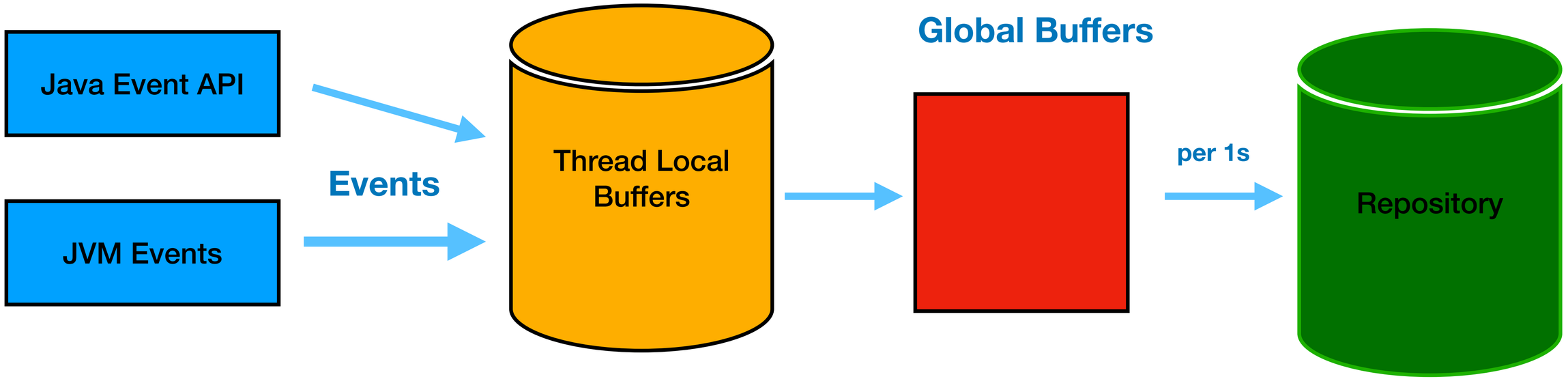
- 1998 Appeal Virtual Machines (AVM) - JRockit JVM
- 2002 AVM acquired by BEA
- 2008 acquired by Oracle
- 2012 JDK 7u4 update: Oracle integrated JFR into the HotSpot
- 2017 JDK 9 : Public APIs for creating and consuming data
- 2018 JDK 11: JMC/JFR announced to be fully Open-Sourced

JFR in bullets

- Java Flight Recorder is an event based tracing framework
- Build directly into the Java Virtual Machine
- Provides access to all internal events
- Allows to create custom events
- Tries to achieve a goal 1% overhead

JFR Under the Hood

Event life cycle



JFR: Event - fundamental element

- Import “**jdk.jfr.Event**”
- Basic Element that caries valuable information

EventID:
Timestamp : when event was taken

Duration: not always

Thread ID: Thread where event has occurred

StackTrace ID: It’s optional referst to the StackTrace, default depth 64

Payload: custom information


```
Import jdk.jfr.Event;

public class SampleEvent extends Event {
    //internal logic
    String message;
}
```

```
…
void someAdvanceLogic() {

        SampleEvent e = new SampleEvent();

        e.message = “Important Information”;
        e.begin();

        // advanced logic

        e.end();
        e.commit();
}
…
```

Event: Tuning up

```
import jdk.jfr.Event
import jdk.jfr.Label
import jdk.jdf.Name

@Name("com.openvalue.events.SampleEvent")
@Label("Sample Event")
class SampleEvent extends Event {

    @Label("Message")
    String name;

    @Label("Value")
    int value;
}
```

Annotations: https://docs.oracle.com/en/java/javase/16/docs/api/jdk.jfr/jdk/jfr/class-use/MetadataDefinition.html

JFR: Performance

- Usage of Thread Local Buffers
- Java Platform Optimization
- Methods: *INLINING*, CODE *ELIMINATION*, *SCALARIZATION*

- What happens when event is enabled / disabled


```
void someAdvanceLogic() {
    SampleEvent e = new SampleEvent();
    e.message = “Important Information”;

    e.begin();

    // advanced logic

    e.commit();
}
```

```
void commit() {
    // IF it’s not enabled -> NOT INTERESTING
    if(isEnabled()){
        //now() reads CPU clock register, cheap check
        long duration = now() - startTime;
        if(duration > THRESHOLD) {
            if (shouldCommit()) {
                // Cheap - Thread local writes
                actuallyCommit();
            }
        }
    }
```

Mikeal Vidstedt presented quite neat pseudo-code that helps to understand to the commit()

JFR: Enabled

```
void someAdvanceLogic(){
    // allocating event
    SampleEvent e = new SampleEvent();

    e.begin();   -> INLINING => e.startTime = now(); ->  e.startTime = <JVM intrinsic>

    // advanced logic

    // timestamp, likewise INLINING, implicit end()
    e.commit();

    // JFR ENABLED STATE
    if(e.isEnabled()){
        // perform additional checks and maybe actuallyCommit()
    }
}
```

JFR: Disabled - part 1

```
void someAdvanceLogic() {
    SampleEvent e = new SampleEvent();

    // INLINING from the previous slide
    e.startTime = <JVM intrinsic>;

    // advanced logic

    //INLINING
    if(false) {          // result e.isEnabled()
        //perform additional checks
        //CODE ELIMINATION -> will be removed
    }
}
```

JFR: Disabled - part 2

```
void someAdvanceLogic() {

    SampleEvent e = new SampleEvent();  //  SCALARIZATION  -> REMOVAL


    e.begin()
     1. initial state:     e.begin();
     2. INLINING =>  e.startTime = <JVM intrinsic>;
     3. INLINING =>  long startTime = <JVM intrisince>;
     4. CODE ELIMINATION => long startTime = <JVM intrisince>;  REMOVAL

    //business logic

}
```

JFR: Disabled - part 3

```
void someAdvanceLogic() {
        //business logic
}
```

JFR: Data Visualistion

OPEN VALUE

- Command line tool avaliable from JDK 11 => **jfr**

```
$jfr summary <JFR_file>
$jfr print –json <JFR_FILE>
```

- JFR GUI. (**DEMO**)
  - Automated analysis
  - Java Application => Thread, Memory, etc.
  - Event Browser

Java Mission Control Project

Upcoming release 8.1

Release	Milestone	Date
==========	==========	==========
8.1.0	RDS	2021-06-02
8.1.0	RDS2	2021-07-02
8.1.0	GA	2021-08-02

- New Allocation Events for JDK 16
- JMC Agent, JMC Agent Plugin (**DEMO**)
- Performance Improvements : Perser, Rules
- Many others

JMC Agent Plugin: preview - 1

- Agent can be attached into the JVM process over the JMX
- Agent is a bytecode transformer
- Agent allows to add JFR instrumentation declaratively (XML)

- Agent Plugin: configuration, information about the transformation and more

- Simple Demo: **VehicleFactories producing Vehicle**

https://developers.redhat.com/blog/2020/10/29/collect-jdk-flight-recorder-events-at-runtime-with-jmc-agent#introduction_to_the_jmc_agent_plugin

JMC Agent Plugin: preview - 2

```
<jfragent>
    <events>
        <event id="vehicle_factory.prepareParts">
        <label>Vehicle Factory Prepare Parts</label>
        <description>Vehicle Factory tries to prepare parts</description>
        <class>com.wengnerits.jfr.agent.demo.VehicleFactory</class>
        <path>demo/vehicle_factory</path>
        <stacktrace>true</stacktrace>
        <method>
            <name>prepareParts</name>
            <descriptor>(Ljava/lang/String;)V</descriptor>
            <parameters>
                <parameter index="0">
                    <name>Preparing Parts</name>
                    <description>factory tries to prepare parts</description>
                </parameter>
            </parameters>
        </method>
    </event>
</jfragent>
```

JMC Agent Plugin: preview - 3

```
@Label("Vehicle Factory Prepare Parts")
@Description("Vehicle Factory tries to prepare parts")
@Category("demo","vehicle_factory")
@StackTrace(true)
public class __JFREventVehicleFactoryPrepareParts extends Event {
    public __JFREventVehicleFactoryPrepareParts () {

    }
}

void prepareParts(String p) throws InterruptedException {
        __JFREventVehicleFactoryPrepareParts e = new __JFREventVehicleFactoryPrepareParts();
        e.begin();
        System.out.println("factory: " + name + ", vehicle: "+ counter +", preparing parts: " + p);
        makeWork();
        state = State.PRODUCING;
        e.commit();
    }
```

JFR: How to Get

- **Clone**: https://github.com/openjdk/jmc. => script **build.sh**
- **AdoptOpenJDK:** http://adoptopenjdk.net/jmc
- **Azul**: https://www.azul.com/products/zulu-mission-control
- **RedHat**: distributes as RPMs in Fedora and RHEL
- **Oracle**: https://www.oracle.com/java/technologies/jdk-mission-control.html

JFR-Tutorial: https://github.com/thegreystone/jmc-tutorial

OPEN VALUE

Q / A

Thank YOU !