# JBoss Community

# Getting into the driver's seat of your JVM

# About me

- Hardy Ferentschik - hardy@hibernate.org
- Hibernate team member
- Project lead of Hibernate Validator
- Bean Validation expert group member
- Interested in everything new and cool, e.g. Openshift, Docker

# All online!

https://github.com/hferentschik/control-your-jvm

user:hferentschik

```
1.8.0_11/Contents/Home/bin]$ ls
appletviewer*   jcmd*           jsadebugd*      rmid*
extcheck*       jconsole*       jstack*         rmiregistry*
idlj*           jdb*            jstat*          schemagen*
jar*            jdeps*          jstatd*         serialver*
jarsigner*      jhat*           jvisualvm*      servertool*
java*           jinfo*          keytool*        tnameserv*
javac*          jjs*            native2ascii*   unpack200*
javadoc*        jmap*           orbd*           wsgen*
javafxpackager* jmc*            pack200*        wsimport*
javah*          jps*            policytool*     xjc*
javap*          jrunscript*     rmic*
```

```
1.8.0_11/Contents/Home/bin]$ ls
                    jcmd*              jsadebugd*
                                       jstack*
                                       jstat*
                                       jstatd*
                    jhat*
                    jinfo*

                    jmap*
                    jmc*
                    jps*
```

“This utility is unsupported and may or may not be available in future versions of the J2SE SDK”

# System View

# Major hardware subsystems

- Processors (CPUs)
- Memory (physical RAM)
- Disk I/O
- Network I/O

# Ask yourself

- How busy are my CPUs?
- Does the system have sufficient memory?
- How much disk I/O is the system generating?
- How much network I/O is occurring?

# System tools

- top, vmstat, sar, iostat, netstat, tcpdump
  - `vmstat -a -S M 5 10`
  - `iostat -x 2 10`

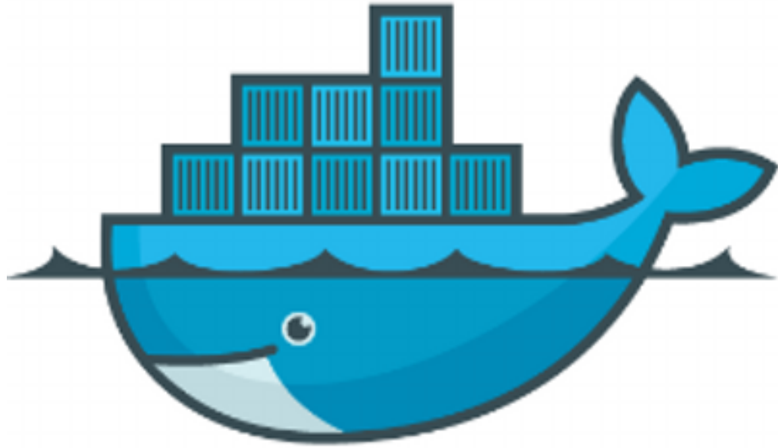
Mac

- top, sar
  - sar -d 5 10
- dtrace, iosnoop, iotop (page_faults.d, sock_j.d, soconnect.d)

# Demo

# Docker container for demos

- http://docker.io
- build, ship, and run apps
- boot2docker for Mac
- make examples re-producible
- simulate a remote access



# Process status

- `jps` - Java Virtual Machine Process Status Tool
  - useful options: `jps -lmv`
- uses the java launcher to find the class name and arguments passed to the main method

# jinfo

```
[hardy@Sarmakand-2 ~ (master)]$ jinfo 22172
Attaching to process ID 22172, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.20-b23
Java System Properties:

java.runtime.name = Java(TM) SE Runtime Environment
java.vm.version = 25.20-b23
sun.boot.library.path = /Library/Java/JavaVirtualMachines/jdk1.8.0_20.j
user.country.format = SE
gopherProxySet = false
java.vendor.url = http://java.oracle.com/
java.vm.vendor = Oracle Corporation
path.separator = :
guice.disable.misplaced.annotation.check = true
file.encoding.pkg = sun.io
java.vm.name = Java HotSpot(TM) 64-Bit Server VM
sun.os.patch.level = unknown
sun.java.launcher = SUN_STANDARD
user.country = US
user.dir = /Users/hardy/work/hibernate/git/beanvalidation/validator
java.vm.specification.name = Java Virtual Machine Specification
java.runtime.version = 1.8.0_20-b26
java.awt.graphicsenv = sun.awt.CGraphicsEnvironment
os.arch = x86_64
java.endorsed.dirs = /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/
line.separator =
```

- Quick overview of used (system) properties of running JVM
- Requires `jsadebugd` for remote execution
- Allows to change *manageable* flags
  - `HotSpotDiagnosticMXBean` (programmatically)
  - `com.sun.management:type=HotSpotDiagnostic` (MBean)

# Thread dumps

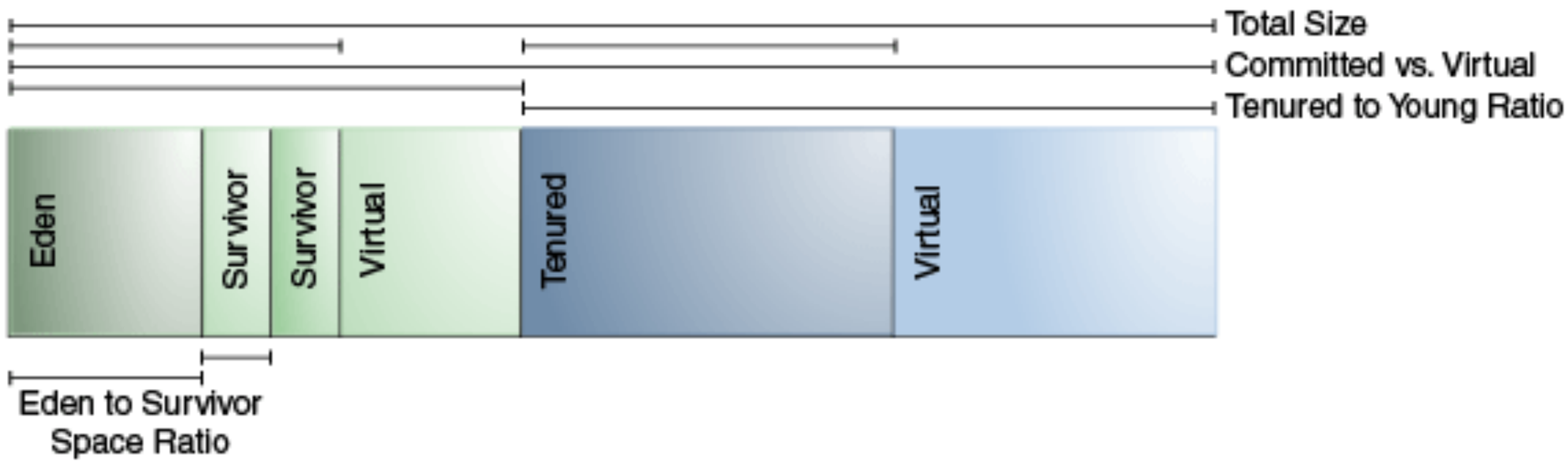
- `jstack <pid>`
- `jcmd <pid> Thread.print`
- `kill -QUIT <pid>`
- `ctrl-\ (control-break handler)`

# Process statistics

- `jstat` - Java Virtual Machine statistics monitoring tool
  - `jstat -options // list all available reports`
  - `jstat -gc <vmid> 2s 10 // 10 GC reports in 2s interval`
- `jstatd` - Virtual Machine jstat Daemon to execute jstat commands remotely

# Demo

# How to read the `jstat` output?




- Understand garbage collection (see links)
- Know your app

BUG

```java
public final class ReflectionHelper {

    private static final String PACKAGE_SEPARATOR = ".";
    private static final String ARRAY_CLASS_NAME_PREFIX = "[L";
    private static final String ARRAY_CLASS_NAME_SUFFIX = ";";

    private static final String PROPERTY_ACCESSOR_PREFIX_GET = "get";
    private static final String PROPERTY_ACCESSOR_PREFIX_IS = "is";
    private static final String PROPERTY_ACCESSOR_PREFIX_HAS = "has";
    private static final String[] PROPERTY_ACCESSOR_PREFIXES = {
            PROPERTY_ACCESSOR_PREFIX_GET,
            PROPERTY_ACCESSOR_PREFIX_IS,
            PROPERTY_ACCESSOR_PREFIX_HAS
    };

    private static final Log log = LoggerFactory.make();

    /**
     * Used for resolving type parameters. Thread-safe.
     */
    private static final TypeResolver typeResolver = new TypeResolver();

    private static final Map<String, Class<?>> PRIMITIVE_NAME_TO_PRIMITIVE;
```

```java
public final class ReflectionHelper {

    private static final String PACKAGE_SEPARATOR = ".";
    private static final String ARRAY_CLASS_NAME_PREFIX = "[L";
    private static final String ARRAY_CLASS_NAME_SUFFIX = ";";

    private static final String PROPERTY_ACCESSOR_PREFIX_GET = "get";
    private static final String PROPERTY_ACCESSOR_PREFIX_IS = "is";
    private static final String PROPERTY_ACCESSOR_PREFIX_HAS = "has";
    private static final String[] PROPERTY_ACCESSOR_PREFIXES = {
            PROPERTY_ACCESSOR_PREFIX_GET,
            PROPERTY_ACCESSOR_PREFIX_IS,
            PROPERTY_ACCESSOR_PREFIX_HAS
    };

    private static final Log log = LoggerFactory.make();

    /**
     * Used for resolving type parameters. Thread-safe.
     */
    private static final TypeResolver typeResolver = new TypeResolver();

    private static final Map<String, Class<?>> PRIMITIVE_NAME_TO_PRIMITIVE;
```

# `jmap` & `jhat`

- `jmap` - prints shared object memory maps or heap memory details
  - Create dump file:

```
jmap -F -dump:format=b,file=heap.bin <pid>
```

- `jhat` - Java Heap Analysis Tool (default port 7000)
  - `jhat heap.bin`
  - `jhat -baseline heap1.bin heap2.bin`

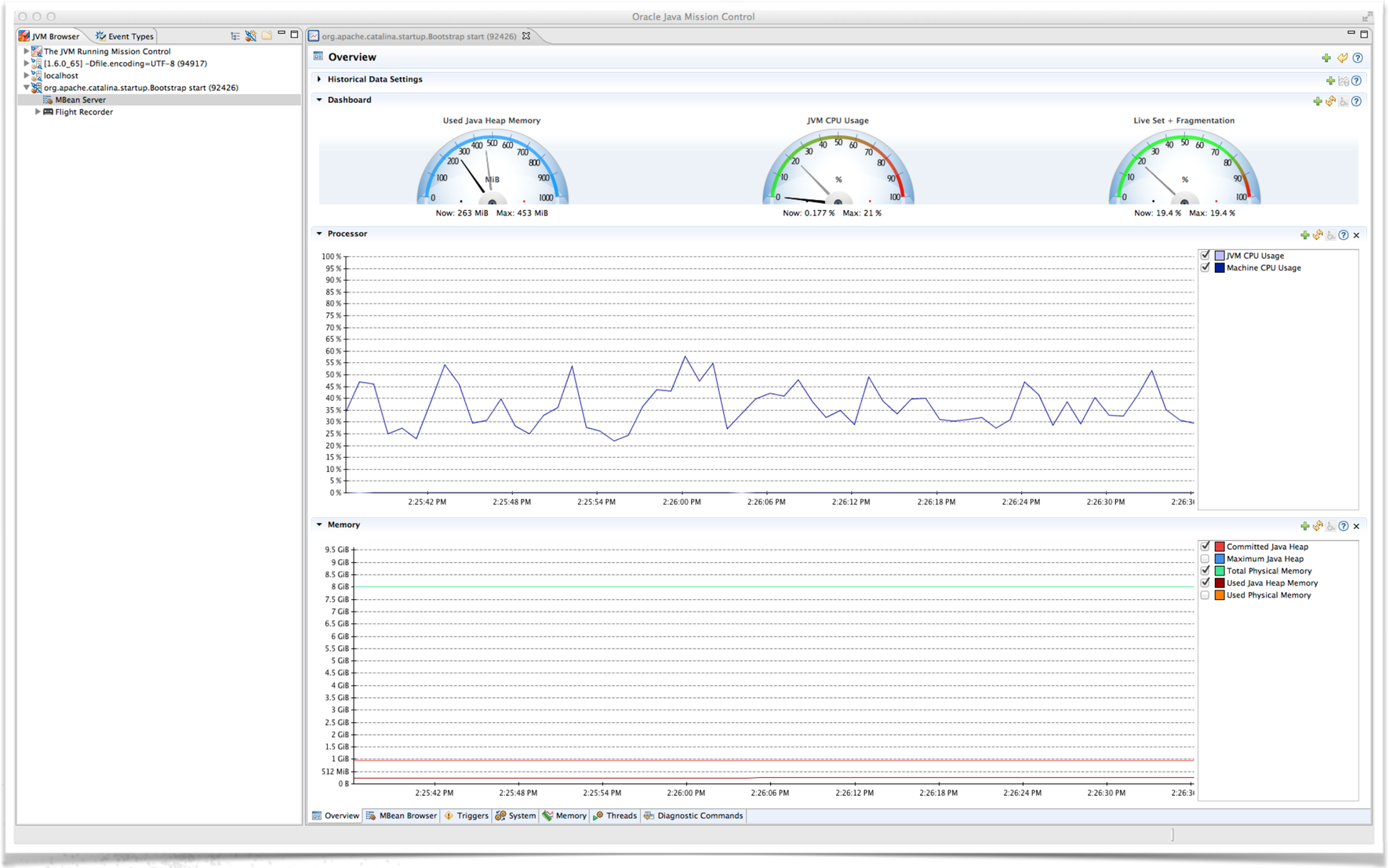
# OQL - Object Query Language

- SQL-like query language to query Java heap

```
select <expression>
[ from [instanceof] <class name>
<identifier>
[ where <filter-expression> ] ]
```

- Based on JavaScript expression language
- Execute query: http://localhost:7000/oql/
- Built-in help: http://localhost:7000/oqlhelp

# Java Mission (JMC)



# Java Mission Control & Flight Recorder

- Part of the Hotspot JDK since Java 7 update (7u40)
- Latest release 5.4.0 as part of Java 8u20
- Roots in the JRockit JVM tooling
- ~ 2% overhead (often less)
- Requires -XX:+UnlockCommercialFeatures and -XX:+FlightRecorder
- http://hirt.se/blog/

# Start recordings (3 options)

- Via the Java Mission Control GUI
- Command line
  - `java -XX:StartFlightRecording=delay=20s,duration=60s,name=MyRecording,filename=/tmp/my-recording.jfr,settings=profile`
- jcmd
  - `jcmd <pid> JFR.start name=MyRecording settings=default`

# Demo

# Tips and Tricks

- Multiple recordings can run in parallel
- The method profiler will not show you time spent in native
- Extendable via plugins, eg DTrace, JavaFX
- Operative sets allow to narrow down events
- Possible to write own Flight Recorder events

# Links

- http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/index.html
- GC explained - http://stackoverflow.com/questions/13660871/jvm-garbage-collection-in-young-generation/13661014#13661014
- Querying Java heap with OQL  - https://blogs.oracle.com/sundararajan/entry/querying_java_heap_with_oql
- Flight Recorder presentation - http://vimeo.com/98922031
- More links - https://github.com/hferentschik/control-your-jvm/

# Q + A