JAVA PERFORMANCE FOR DEVELOPERS:

HOW TO GET STARTED, PERFORM ANALYSIS AND AVOID COMMON MISTAKES

Alessio Soldano

alessio.soldano@jboss.com

Principal Software Engineer

JBoss - Red Hat

November 17th, 2016

Java performance for developers: how to...
By Alessio Soldano

Present

PERFORMANCE TUNING

- System optimization
- GC log analysis...
- JVM tuning (Xmx, MaxPermSize, MaxMetaspaceSize, ...)
- ...

Present

PERFORMANCE TUNING

- System optimization
- GC log analysis...
- JVM tuning (Xmx, MaxPermSize, MaxMetaspaceSize, ...)
- ...

...SOMEONE ELSE WILL TAKE CARE OF THIS!

Present

THE PERFORMANCE OF AN APPLICATION DEPENDS ON HOW WELL IT IS WRITTEN!

Present

ALGORITHMS

- Time complexity O(n)
- Choosing proper data structures

Present

SMALL IS BETTER

- Less code to compile (JIT), less time before optimum perfomances
- Less objects, less work for GC
- Less survivor objects, shorter GC cycles
- Less classes to load from disk, better boot time ;-)
- ...

Present

HAVE YOUR APPLICATION DO ALL YOU REALLY NEED AND NOTHING MORE!

Present

GENERAL OPTIMIZATION TIPS

Present

EARLY STAGES OF DEV CYCLES

- Premature optimization?
- Avoid common / stupid mistakes

Present

COMMON CASES

Edge usecases deserve less attention
compared to common ones.

Present

BIG FISHES FIRST

Focus on the code that's consuming more resources (cpu / memory) first

Present

OCCAM'S RAZOR

The simplest explanation for a performance issue is the most conceivable cause.

Present

COMMUNICATING VESSELS

Fully understanding the analysed system/application architecture is required to figure out whether an optimization is actually helping or not.

Enhancing a component might even change system resources allocation in a way that actually gives worse overall performances.

Present

TEST THE ENTIRE APPLICATION!

Do not rely on components' micro-benchmarks only. What you really care is the performance of the application as a whole.

Present

TESTING & EVALUATING RESULTS

Present

WHAT ARE WE INTERESTED IN?

- Throughput vs response time (avg / min / max)
  - tradeoff
  - pooling
- Runtime vs boot performances
  - GC effects

Present

WARM-UP RELEVANCE

- JIT compilation
- Cache
- ...

Present

INTERPRETATION OF RESULTS

- Variability
- Multiple tests
- Statistical significance

Present

TESTING CONDITIONS

Present

LOOK AT CPU USAGE FIRST

- We want it as high as possible
  - why it isn't 100%?
  - lock?
  - bottleneck I/O (disk, network, ...)?
- Isolate load generator from system under test
- GC effects

Present

FIGURE OUT PROPER LOAD TO TEST

- should be representative of actual production usage
- ... but still allowing performing analysis

Present

TOOLS

Present

HEAP HISTOGRAMS

```
alessio@localhost ~ $ jmap -histo:live 16620

 num     #instances         #bytes  class name
----------------------------------------------
   1:         94968        9202416  [C
   2:        100057        3201824  java.util.HashMap$Node
   3:         73898        3065976  [Ljava.lang.Object;
   4:         94800        2275200  java.lang.String
   5:         13364        1517112  java.lang.Class
   6:         62031        1488744  java.util.ArrayList
   7:          6809        1343744  [I
   8:         13006        1334520  [Ljava.util.HashMap$Node;
   9:         14208         681984  java.util.HashMap
  10:          7016         673536  java.util.jar.JarFile$JarFileEntry
  11:          2149         487416  [B
  12:          4778         420464  java.lang.reflect.Method
  13:          9857         394280  java.util.LinkedHashMap$Entry
  14:          7933         253856  java.util.RegularEnumSet
  15:          7514         240448  org.jboss.vfs.spi.JavaZipFileSystem$ZipNode
  16:          6493         207776  java.util.Hashtable$Entry
  17:          8569         205656  java.util.jar.Attributes$Name
  18:          5217         166944  java.util.concurrent.ConcurrentHashMap$Node
  19:          1621         155616  org.jboss.as.controller.SimpleAttributeDefinition
  20:          4807         155352  [Ljava.lang.String;
```

- Memory consumed by **objects grouped by class**
- **Shallow** object size
- *jmap -histo <pid>* to skip full GC before count (includes objects eligible to be collected)
- *jcmd <pid> GC.class_histogram* to count live object only (but it does not force full GC)
- The count is often worthless due to objects being shared and linked to each others...

Present

HEAP DUMPS

- Captured using *jmap -dump:live,file=dump.hprof <pid>*
- Misc analysis tools (e.g. Eclipse **M**emory **A**nalyzer **T**ool)
- Allow for **retained** object size analysis (includes uniquely referenced objects) and **deep** object size analysis (includes any referenced object, even if shared b

Overview | Histogram | dominator_tree

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.HashMap	14.063	675.024	>= 15.722.384
java.util.HashMap$Node[]	12.573	1.309.224	>= 15.636.248
java.util.HashMap$Node	99.895	3.196.640	>= 14.462.952
java.lang.String	87.287	2.094.888	>= 10.495.200
char[]	87.446	8.554.016	>= 8.554.016
java.lang.Object[]	67.400	2.720.864	>= 5.823.568
java.util.ArrayList	61.689	1.480.536	>= 5.162.560
byte[]	6.076	5.132.280	>= 5.132.280
org.jboss.modules.Module	265	12.720	>= 5.088.832
org.jboss.modules.Linkage	266	8.512	>= 5.012.544
java.lang.ref.Finalizer	4.726	189.040	>= 5.005.936
java.util.zip.ZipFile$ZipFileIr	1.364	76.384	>= 4.813.696
java.lang.Class	12.970	126.176	>= 3.912.104
org.jboss.modules.ModuleC	265	23.320	>= 3.748.192

Present

HEAP DUMPS

- Allow analysis of dominator object tree, perhaps ordered by retained memory:

Overview | Histogram | dominator_tree

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.jar.JarFile @ 0xe06d6188	64	1.586.616	4,72%
org.jboss.as.controller.registry.NodeSubregistry @	32	954.928	2,84%
org.jboss.metadata.web.spec.Tld21MetaData @	72	569.232	1,69%
class sun.security.jca.Providers @ 0xe030b1f0 Sy	24	451.752	1,34%
java.util.jar.JarFile @ 0xe06d56f8	64	391.928	1,17%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe19a10e	40	382.416	1,14%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe18f853	40	338.656	1,01%
class java.util.ResourceBundle @ 0xe0328cd8 Sy	24	301.304	0,90%
org.jboss.as.jaxrs.deployment.JaxrsSpringProcess	24	294.400	0,88%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe15d5fc8	40	267.504	0,80%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe16e4fc8	40	261.512	0,78%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe183252	40	254.656	0,76%
class java.beans.ThreadGroupContext @ 0xe1f43	8	252.168	0,75%
org.jboss.as.controller.registry.ConcreteResourceR	80	226.768	0,67%
org.jboss.vfs.spi.JavaZipFileSystem @ 0xe18a8c9	40	224.624	0,67%
org.infinispan.manager.DefaultCacheManager @	40	223.936	0,67%
org.jboss.modules.ModuleClassLoader @ 0xe0b2	88	218.848	0,65%
org.infinispan.manager.DefaultCacheManager @	40	215.000	0,64%

Present

HEAP DUMPS

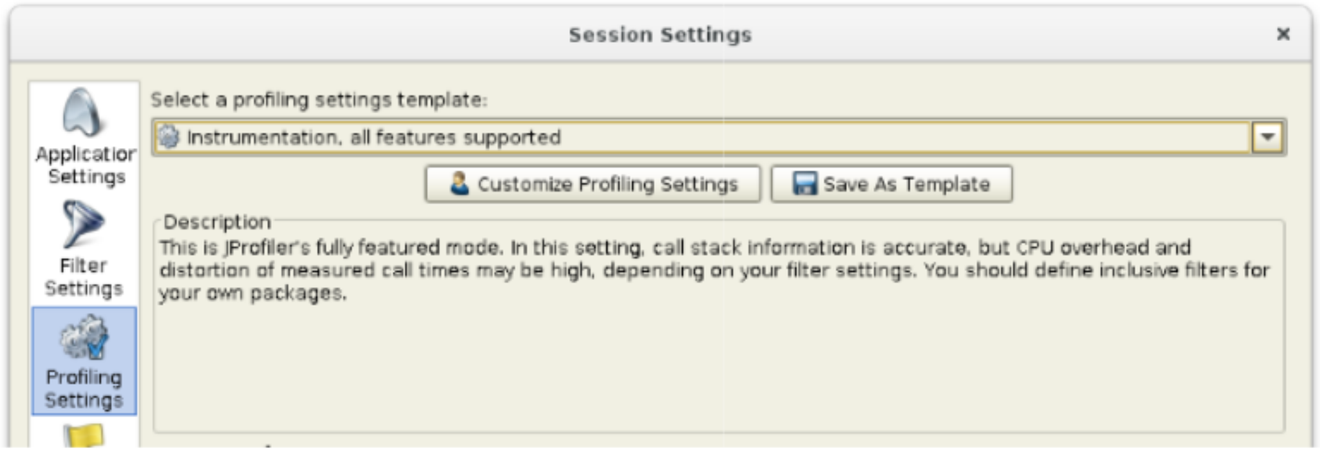
- ... and checking back traces of object references (GC roots):

Overview | Histogram | dominator_tree | list_objects [selection of 'Tld21MetaData @ 0xe1042870'] -inbound

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
org.jboss.metadata.web.spec.Tld21MetaData @ 0xe1042870	72	569.232
[0] java.lang.Object[10] @ 0xe1006ae8	56	56
elementData java.util.ArrayList @ 0xe1006ad0	24	80
value java.util.HashMap$Node @ 0xe1006ab0	32	112
[10] java.util.HashMap$Node[16] @ 0xe1006a60	80	192
table java.util.HashMap @ 0xe1006a30	48	240
jsfTldMap org.jboss.as.jsf.deployment.JSFSharedTldsProcessor @ 0xe1006a20	16	256
processor org.jboss.as.server.deployment.RegisteredDeploymentUnitProcessor @	24	280
[10] java.lang.Object[70] @ 0xe14b9a08	296	10.512
elementData java.util.ArrayList @ 0xe14b99f0	24	10.536
[1] java.lang.Object[9] @ 0xe14b9520	56	22.728
vals java.util.EnumMap @ 0xe14b94f8	40	22.768
phases org.jboss.as.server.deployment.DeployerChains @ 0xe14b94e	16	22.784
[0] java.lang.Object[10] @ 0xe1434840	56	56
[15] java.lang.Object[22] @ 0xe15745f0	104	104
Σ **Total: 3 entries**		

Present

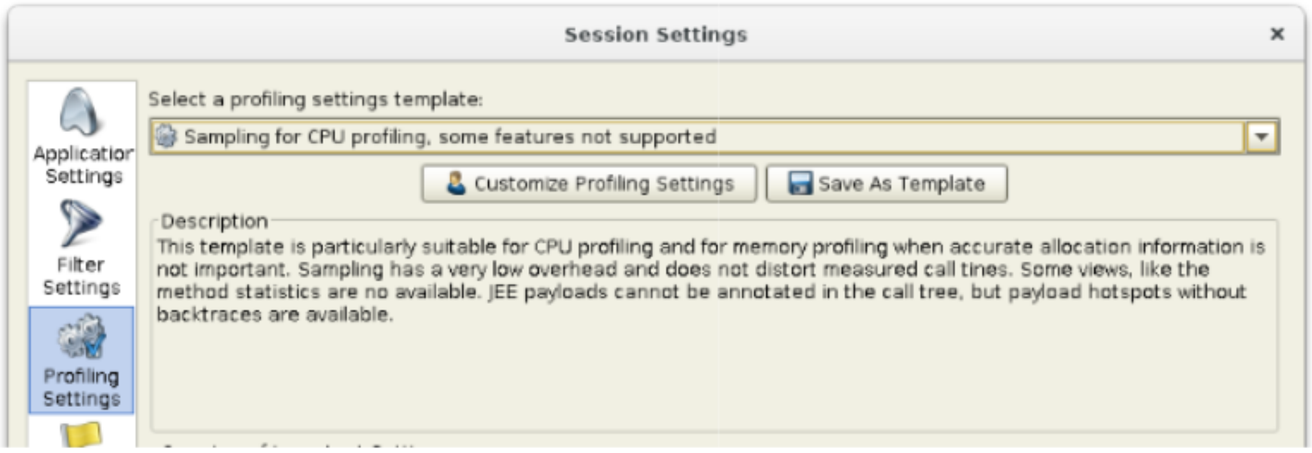
INSTRUMENTED PROFILERS



- Bytecode instrumentation
- Accurate invocation count
- High overhead
- Potentially distorted results (due to high load and effects on JIT compiler)

Present

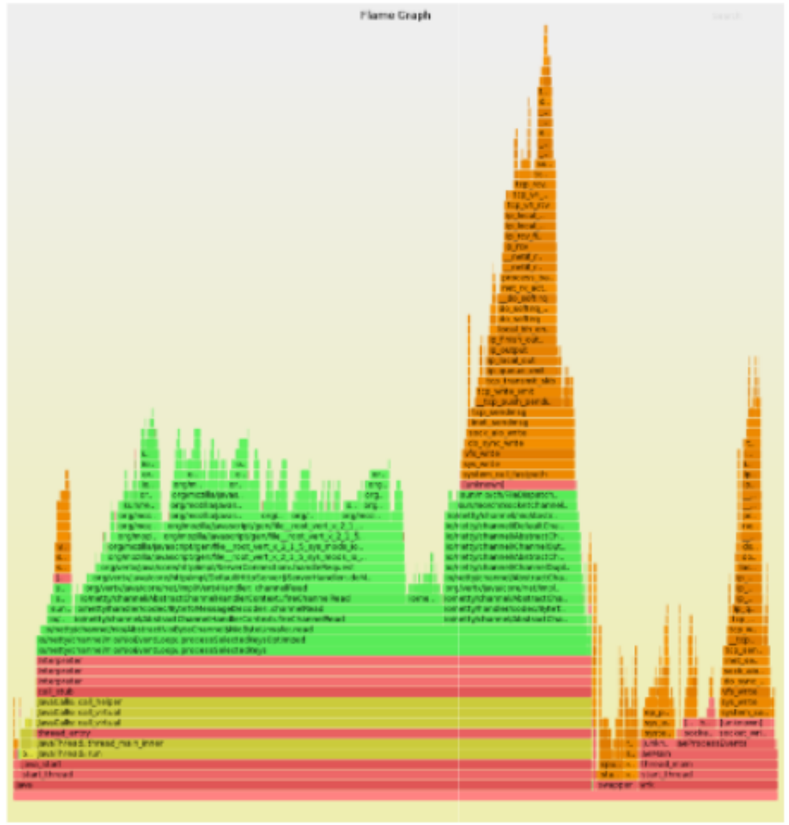
SAMPLING PROFILERS



- Stacktrace sampling
- Low overhead
- Require proper sampling frequency and total recording period to give reliable results
- Safepoints...

Present

CPU FLAME GRAPHS



- Stacktrace sampling
- Compact / interactive view (SVG+JS)
- JVM and system informations at the same time

Present

DEMO

How to look at a flame graph

10.9

Present

CPU FLAME GRAPHS

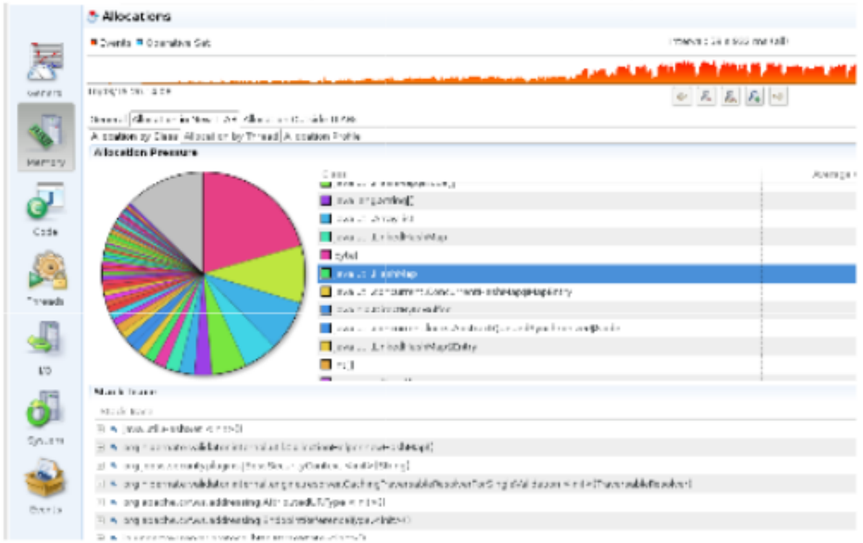
generated as follows on Linux:

- *perf* for data sampling
- *perf-map-agent* for Java symbol tables generation
- *FlameGraph.pl* to create the svg file
- *JDK 1.8 u60 or greater* required to preserve RBP registry (*-XX:+PreserveFramePointer*)

Present

JAVA MISSION CONTROL / FLIGHT RECORDER

- Included in Oracle JDK (*jmc*)
- Based on JVM events
- 70+ event types, covering classloading, thread statistics, exceptions, memory / file / socket / lock / monitor usage, JIT compiling and GC activity
- Low overhead (potentially usable in production too)



Present

DEMO

- How to start a JFR recording (event setup, ...)
- Brief look at the available reports

Present

CODE EXAMPLES

Present

STRINGS

- String is immutable
- lot of time spent on building and merging them (memory arrays allocation)
- HotSpot automatically converts "+" into *StringBuilder::append()*

```
int n = 4;

String cn = "a" + n + "b" + n; //OK

StringBuilder sb1 = new StringBuilder();
sb1.append("a");
sb1.append(n);
sb1.append("b");
sb1.append(n);
String cn1 = sb1.toString(); //OK

StringBuilder sb2 = new StringBuilder();
sb2.append("a" + n); //NO!!
sb2.append("b" + n); //NO!!
String cn2 = sb2.toString();
```

```
String dn = "a" + n + (n > 3 ? "b" + n : "B" + n/2); //NO!!!

StringBuilder sb3 = new StringBuilder();
sb3.append("a");
sb3.append(n);
if (n > 3) {
   sb3.append("b");
   sb3.append(n);
} else {
   sb3.append("B");
   sb3.append(n/2);
}
String dn1 = sb3.toString(); //OK
```


Present

...AND LOGS

- Do not concatenate strings to build up messages that might not be actually logged due to logger's level setup
- *isLoggable(..)* already internally invoked on each logger call...

```
LOG.debug("foo");  //OK
if (LOG.isLoggable(Level.FINE)) {
   LOG.debug("foo"); //NO
}

LOG.debug("foo " + n); //NO
if (LOG.isLoggable(Level.FINE)) {
   LOG.debug("foo " + n); //OK
}
```

- Consider using logging API with argument placeholders

Present

PATTERNS

- Avoid **multiple** *String.replace(..)*, *String.replaceAll(..)*, *String.split(..)*
- Compile frequently used patterns in advance and reuse

```
    29  +import java.util.regex.Pattern;
29  30
30  31   public final class HttpHeaderHelper {
        @@ -44,6 +45,7 @@
44  45
45  46       private static Map<String, String> internalHeaders = new HashMap<St
46  47       private static ConcurrentHashMap<String, String> encodings = new Co
    48  +    private static Pattern charsetPattern = Pattern.compile("\"|'");
47  49
        @@ -110,8 +112,7 @@ public static String mapCharset(String enc, String c
110 112           }
111 113           // Charsets can be quoted. But it's quite certain that they car
112 114           // anything like that.
113     -         enc = enc.replace("\"", "").trim();
114     -         enc = enc.replace("'", "");
    115 +         enc = charsetPattern.matcher(enc).replaceAll("").trim();
```


Present

REFLECTION

- Reflection is generally time consuming

```
77  77      private Class<?>[] getParameterTypes(Method method, Object[] args) {
78  78          Class<?>[] types = method.getParameterTypes();
    79          final Annotation[][] parAnnotations = method.getParameterAnnotations();
79  80          for (int x = 0; x < types.length; x++) {
80              UnwrapParam p = getUnwrapParam(method.getParameterAnnotations()[x]);
    81          UnwrapParam p = getUnwrapParam(parAnnotations[x]);
81  82              if (p != null) {
```


Present

COLLECTIONS

- Avoid useless copies of collection instances
- Rather enhance your API to support additional usecase...

el/AbstractMessageContainer.java (+15 -0)

```
    public List<MessagePartInfo> getMessageParts() {
        if (outOfBandParts == null) {
            return new ArrayList<MessagePartInfo>(messageParts.values());
        }
        List<MessagePartInfo> parts = new ArrayList<MessagePartInfo>(messageParts.values());
        parts.addAll(outOfBandParts);
        return parts;
    }
    public int getMessagePartsNumber() {
        if (outOfBandParts == null) {
            return messageParts.size();
        }
        return messageParts.size() + outOfBandParts.size();
    }
    public MessagePartInfo getFirstMessagePart() {
        if (!messageParts.isEmpty()) {
            return messageParts.values().iterator().next();
        } else  if (outOfBandParts != null && !outOfBandParts.isEmpty()) {
            return outOfBandParts.get(0);
        } else {
            return null;
        }
    }
    public List<MessagePartInfo> getOutOfBandParts() {
        if (outOfBandParts == null) {
            return Collections.emptyList();
        }
        return Collections.unmodifiableList(outOfBandParts);
    }
```


Present

COLLECTIONS

3 ...ndings/soap/src/main/java/org/apache/cxf/binding/soap/interceptor/SoapOutInterceptor.java

```
@@ -214,8 +214,7 @@ private boolean handleHeaderPart(boolean preexistingHeaders, SoapMessage
214 214             return endedHeader;
215 215         }
216 216
217     -       List<MessagePartInfo> parts = wrappedBmi.getMessageInfo().getMessageParts();
218     -       if (parts.size() > 0) {
    217 +       if (wrappedBmi.getMessageInfo().getMessagePartsNumber() > 0) {
219 218             MessageContentsList objs = MessageContentsList.getContentsList(message);
220 219             if (objs == null) {
221 220                 return endedHeader;
```

8 rt/bindings/xml/src/main/java/org/apache/cxf/binding/xml/XMLFormatValidator.java

```
@@ -67,8 +67,8 @@ private boolean checkXMLFormat(BindingInfo binding) {
67 67          for (BindingOperationInfo bo : bos) {
68 68              OperationInfo op = binding.getInterface().getOperation(bo.getName());
69 69              needRootNode = false;
70     -           if (op.getInput().getMessageParts().size() == 0
71     -               || op.getInput().getMessageParts().size() > 1) {
    70 +           final int inputPartsNum = op.getInput().getMessagePartsNumber();
    71 +           if (inputPartsNum == 0 || inputPartsNum > 1) {
72 72                  needRootNode = true;
73 73              }
74 74              if (needRootNode) {
```


Present

COLLECTIONS

- Consider lazy creation
- Choose proper implementation! (is ordering relevant? are multiple threads conccurently using the collection? ...)
- Properly set initial size, if possible
  - re-sizing usually means re-allotation
  - ...but beware of wasted memory
- Beware of leaks: cleanup or get rid of references to collections that are not used anymore!

Present

CLASSLOADERS

Beware of **memory leaks** (the classloader holds a ref to every class it loads); so, e.g. if a classloader is not needed anymore and meant to be deleted:

- do not leave it set as thread context classloader for any thread
- do not leave it directly referenced in any static cache or attribute...

Present

URL

- Building up a URL is expensive
  - a lot of String processing
  - URLStreamHandler
- Do not keep on converting between "equivalent" representations of addresses (URL, URI, String)

Present

EXCEPTIONS

- *stacktraces* are expensive
- Use exceptions for exceptional conditions
- Replace them with proper result types
- Override Throwable.fillInStackTrace()

Present

THANK YOU :-)

Present