

INSANE: Java Application Heap Postmortem Analysis In Practice

Radim Kubacki
Tonda Nebuzelsky

Sun Microsystems - NetBeans
<http://performance.netbeans.org>
BOF 9195

Goal of The Talk

Learn specifics of memory footprint in Java and learn how to analyze Java heap of an application using INSANE tool

Presentation Agenda

- Specifics of memory footprint in Java
- INSANE – powerful heap analysis tool
- Searching heap for suspicious patterns
- Searching for memory leaks, DEMO
- Using INSANE in tests
- Q&A

Presentation Agenda

- **Specifics of memory footprint in Java**
- INSANE – powerful heap analysis tool
- Searching heap for suspicious patterns
- Searching for memory leaks, DEMO
- Using INSANE in tests
- Q&A

Measuring memory footprint

Overview

- footprint
 - virtual size (VSZ) – mem allocated from available space
 - `pmap` on Linux shows the memory consumers for a process
 - `jmap` shows mapping of shared objects (libs) for the process
 - resident size (RSS) – blocks paged in physical memory
 - pages can be shared between processes
 - sum of all RSS can be greater than total used RAM
- examples (on Linux w/ JDK 5)
 - RSS starting at 22 MB for simple Java GUI app, with VSZ around 250 MB
 - typical NetBeans resident size is 120-250 MB, with VSZ around 650MB

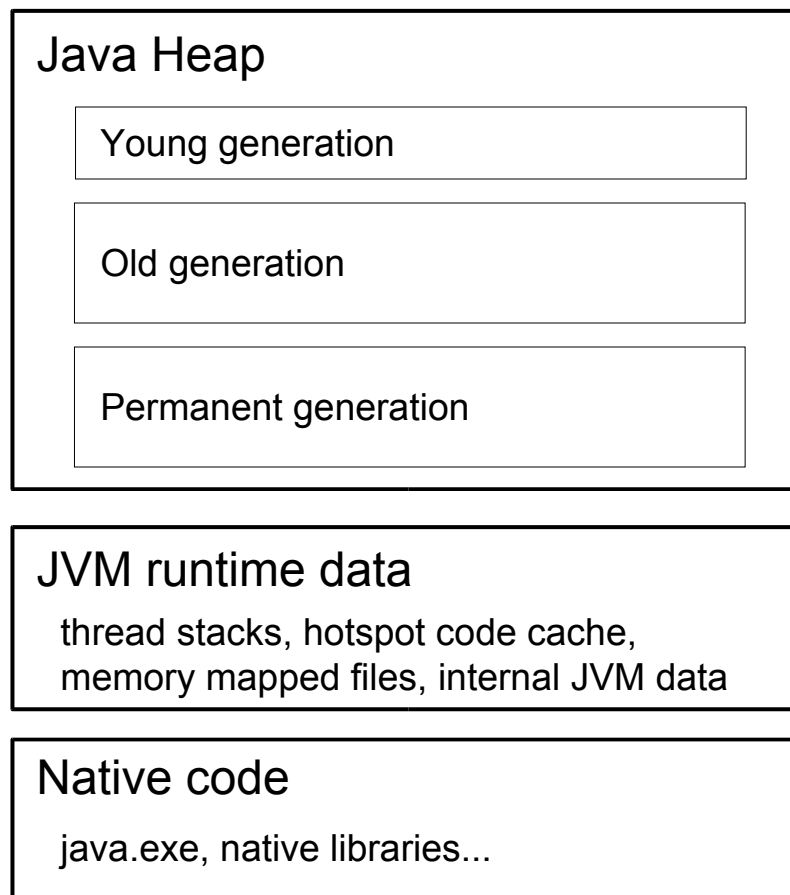
Specifics of memory footprint in Java

Overview

- Native code – java.exe, native libraries
- JVM runtime data
 - thread stacks
 - hotspot code cache
 - memory mapped files
 - internal data
- Java Heap: young+old generations, perm gen.
 - -Xms4m -Xmx64m
 - -Xmn (-XX:NewSize -XX:MaxNewSize)
 - -XX:NewRatio (new/old), -XX:SurvivorRatio (eden/surv)
 - -XX:PermSize=8m -XX:MaxPermSize=64m

Specifics of memory footprint in Java

The picture



Java specific implications

Specifics

- footprint of Java application is usually bigger than with native application
- severe consequences of memory swapping (unfriendly to GC - bad locality of objects)
 - keep size of heap reasonable, don't oversize it
 - be aware that application minimize can provoke so called “working set trimming” – especially on Windows – which will in return affect UI responsiveness at application restore time
 - non-compacting GC algorithms behave worse than the default compacting ones

Classes

Overview

- Class metadata occupy major part of permanent generation area
- Simple GUI app loads >1500 classes
- Tracking
 - `jstat -class`
 - `java -verbose:class`
- Possible improvements
 - exclude debug info (up to -15% of class file size)
 - obfuscation – common practice in J2ME world
 - write less code – reuse, eliminate dead code

Presentation Agenda

- Specifics of memory footprint in Java
- **INSANE – powerful heap analysis tool**
- Searching heap for suspicious patterns
- Searching for memory leaks, DEMO
- Using INSANE in regression tests
- Q&A

INSANE – Heap Analysis Tool

Overview

- Requirements
 - Simple enhancing of your application
 - No application runtime overhead
- Writing the information into XML file
- Postmortem analysis of the heap dump
 - Performing queries and computing stats from the dump
- Invoking INSANE methods from tests
 - to find out / verify a structure size
 - to check for unexpected outgoing references

INSANE – Heap Analysis Tool

How it works

- tracing
 - single tracing engine – BFS
 - reflection to find reference fields
 - filtering out INSANE's own objects and weak references
 - reports found objects and references for direct processing or to be stored in a dump
- finding roots
 - no JVM support
 - static, native and stack references
 - stack references are rare in an idle application
 - system classes, user classes from ClassLoader

Example of INSANE dump format

```
<insane>
...
<object id='1040' type='java.lang.String' size='24' />
<ref to='1040' name='sun.misc.URLClassPath.USER_AGENT_
  JAVA_VERSION' />
...
<object id='16fa' type='[C' size='48' value='UA-Java-
Version' />
<ref from='1040' to='16fa' name='java.lang.String.value' />
...
</insane>
```

```
static String USER_AGENT_JAVA_VERSION = "UA-Java-Version";
```

Presentation Agenda

- Specifics of memory footprint in Java
- INSANE – powerful heap analysis tool
- **Searching heap for suspicious patterns**
- Searching for memory leaks, DEMO
- Using INSANE in regression tests
- Q&A

Searching for suspicious patterns

Distribution of objects

- Most frequent objects on heap
 - char[], String
 - collections
 - arrays
- Overall count of objects and their sizes does not tell us enough
- Complex data structures need to be evaluated and categorized

Name	Instance Count	Size
java.lang.String	139,863	4,077,192
char[]	164,698	13,416,592
java.util.HashMap\$Entry	153,071	3,795,704
com.sun.jmx	73,577	5,168,520
String[]	67,227	1,698,780
java.util.HashMap\$Entry	63,204	2,214,464
int[]	41,005	5,011,960
java.util.HashMap\$KeyIterator	38,335	614,160
java.util.HashMap\$KeyIterator	33,263	1,064,576
java.lang.String	28,728	459,648
java.awt.Rectangle	24,853	598,592
java.util.HashMap	21,407	858,780
byte[]	20,132	1,201,192
java.io.File	13,941	221,000
java.lang.Integer	12,303	198,348
java.util.HashMap\$Entry	11,038	266,112
char[]	10,272	672,576
java.lang.ref.WeakReference	7,129	244,796
org.netbeans.j2se.netbeans.j2se.Monitor	6,655	231,720
java.util.ArrayList	5,012	210,760
java.lang.Character	4,067	121,072
java.util.HashSet	3,928	110,348
java.lang.Class	3,378	561,264
java.ioHeapCharBuffer	3,036	288,728
java.ioHeapByteBuffer	2,750	288,780
sun.java2d.p2d.Region	2,710	187,400
org.openide.util.WeakSet\$WeakSetEntry	2,309	153,940
java.util.HashMap\$Entry	2,210	71,500
java.util.HashMap\$Entry	2,186	68,556
java.util.HashMap\$Entry	2,100	164,000
java.util.Date	2,077	98,048
org.netbeans.j2se.netbeans.j2se.Monitor	1,258	150,720

Searching for suspicious patterns

Part 1 – java.lang.String

- java.lang.String
 - 24 bytes (instance) + 12 bytes (arr. header) + 2*length
- patterns
 - many duplicate Strings on heap
 - `egrep '\[C' | cut -d\' -f8 | sort | uniq -c`
 - implement String pool or use `String.intern()`
 - substrings which hold the original large char[] in memory
 - result of `String.substring()` call
 - should be replaced with `new String()`
 - strings used unnecessarily as HashMap keys
 - use another data structure than String with good `hashCode()`

Searching for suspicious patterns

Part 2 - java.util.HashMap

- java.util.HashMap
 - 40 bytes (instance) + 12 bytes + 4*capacity + 24*size
 - 120 bytes for empty HashMap
- patterns
 - many unused empty hash maps on heap
 - wrongly sized hash maps
 - hash maps with bad distribution of entries, effectively collapsed to a linked list

Searching for suspicious patterns

Part 2 - java.util.HashMap

```
Model model = Support.parseXmlFile("insane-dump.xml");
Collection maps = model.getObjectsOfType
    ("java.util.HashMap");
for (Iterator it = maps.iterator(); it.hasNext(); ) {
    Item itm = (Item)it.next();
    Set reachable = reachableFrom(itm);
    if (reachable.size() <= 2) {
        // the map does not contains any entry
        // and print their root reference chains
        Support.findRoots(model, itm, false);
    }
}
```

Searching for suspicious patterns

Part 3 - other

- listeners
 - multiply registered
 - repeatedly added but never removed
 - causes performance overhead
 - leaking
 - too many listeners kept by one `PropertyChangeSupport`

Presentation Agenda

- Specifics of memory footprint in Java
- INSANE – powerful heap analysis tool
- Searching for suspicious patterns on heap
- **Searching for memory leaks, DEMO**
- Using INSANE in regression tests
- Q&A

DEMO

INSANE

Searching for memory leaks

Overview

```
public class MyDialog extends JDialog
    implements PropertyChangeListener {

    public MyDialog () {
        LogManager.getLogManager().
            addPropertyChangeListener(this);
        ...

    public void propertyChange(PropertyChangeEvent evt) {
        ...
    }
}
```

Presentation Agenda

- Specifics of memory footprint in Java
- INSANE – powerful heap analysis tool
- Searching for suspicious patterns on heap
- Searching for memory leaks, DEMO
- **Using INSANE in regression tests**
- Q&A

Using INSANE in regression tests

Overview

- `assertGC (String msg, WeakReference ref)`

```
Object obj = ...;  
WeakReference ref = new WeakReference (obj);  
doSomething();  
obj = null;  
assertGC ("The object can be released", ref);
```

- `assertSize (String msg, long size, Object obj)`

```
class Data {  
    int value;  
}  
Object measure = new Data();  
assertSize ("The object is small", 16, measure);
```


Using INSANE in regression tests

Test output

```
Testcase: testLeak(leakdemo.MyDialogTest): FAILED
junit.framework.AssertionFailedError: Dialog should be released:
private static java.util.logging.LogManager
    java.util.logging.LogManager.manager->
java.util.logging.LogManager@1b994de->
java.beans.PropertyChangeSupport@11c19e6->
sun.awt.EventListenerAggregate@9e0c2d->
[Ljava.beans.PropertyChangeListener;@b524aa->
leakdemo.MyDialog@14d3343
    at org.netbeans.junit.NbTestCase.assertGC
(NbTestCase.java:900)
    at leakdemo.MyDialogTest.testLeak(MyDialogTest.java:32)
```

...

For More Information

- Radim.Kubacki@sun.com
Antonin.Nebuzelsky@sun.com
- <http://performance.netbeans.org/insane>
- <http://openide.netbeans.org/tutorial/test-patterns.html>
- BOF-9956 - Using the Tools in JDK 5.0 to Diagnose Problems and Monitor Applications
- BOF-9937 – Six Ways to Meet OutOfMemoryError

Q&A

Radim Kubacki
Tonda Nebuzelsky

INSANE: Java Application Heap Postmortem Analysis In Practice

Radim Kubacki
Tonda Nebuzelsky

Sun Microsystems - NetBeans
<http://performance.netbeans.org>
BOF 9195