Memory Analysis Simplified
Automated Heap Dump Analysis

for Developers, Testers and Technical Support Employees

Andreas Buchen, SAP
Eclipse Memory Analyzer

- Provide General Purpose Tooling to analyze Java Heap Dumps.
- Report automatically detected Leak Suspects.
- Stable Code Basis donated by SAP.

- Pluggable
  - Heap Formats
  - Application Knowledge
  - Heap Inspections
Agenda

• Automation for Technical Support Staff
  ♦ Leak Suspects
  ♦ Memory Intensive Threads
• Automation for Developers
• Questions & Answers
Support Use Case – Goals

• Find Leaks without immediate Familiarity and Expertise with the Java Code at hand
• Search if the Issue is known (and a Fix available)
• Forward to the responsible Development Support
Support Use Case – Scenario

VM throws an Out Of Memory Error
-XX:+HeapDumpOnOutOfMemoryError

The Heap Dump is a Snapshot of all Objects alive at one Point in Time: objects, classes, class loaders.

Parse & Report Leak Suspects

Developer continues Analysis Interactively
Demo

- Parse Heap Dump
- Investigate Leak Report
Any up-to-date architecture loads components with separate class loaders, be it OSGi or JEE application servers. Extensible to display meaningful names.

Keywords
- One instance of " illuminate:mnr:mmr:lib" occupies 2,266,297,448 (91.26%) of the memory, accumulated in one instance of "java.lang.Object[]" loaded by "<system class loader>".
- Keywords:
  - com.sap.ip:mmr:Repository
  - library:bi:mmr:mmr:lib
  - java.lang.Object[]

Classification for trouble ticket system

Search tuple: Identify if problem is known.
Less ping-pong of trouble tickets.
Shortest Paths to the Accumulation Point

Who is keeping the leak suspect alive?

Accumulation Point in the Dominator Tree

What is kept alive by the leak suspect?
A Simple Object Graph

LinkedList

LinkedList$Entry

header
next
previous

element
String
value
char[]
How to Calculate the Retained Size

The Retained Size of an Object X is the Memory that would be freed by the Garbage Collector if no References to the Object X would exist.
How to Calculate the Retained Size (2)

- To calculate the Retained Size, one removes any References to the object and marks all Objects reachable from the GC Roots. The sum of the Shallow Sizes of the non-marked Objects is the Retained Size.
Garbage Collection (GC) Roots

- GC Roots are objects that are assumed to be reachable. Typically, these include all objects referenced from the current call stacks and classes loaded by the system class loader.
Dominator Tree: Transformation into a “Keep-Alive” Tree
Dominator Tree

• The Dominator Tree is a Transformation of the Cyclic Object Graph into a „Keep-Alive“ Tree:
  - Every Node in the Tree is directly responsible for keeping alive its Children.
  - “X dominates Y if all paths from the roots to Y run through X”

Dominator Tree – Benefits

→ Fast Calculation of the Retained Size (sum all children)
→ List of Distinct Big Objects (first Level of the Tree)
→ Fast Identification of responsible Objects (just go up the tree)

What is a Leak Suspect?

- **Leak Suspect**
- **Dominator Tree**
- **Accumulation Point**

- Top Level Object in Dominator Tree above a given Threshold
- Significant Drop in the Retained Sizes!
Who is Keeping the Leak Suspect Alive?

Shortest Path(s) to GC Roots

Note: All Paths must go through the Parent Nodes, but not necessarily directly (see Linked List Example).
What Is Kept Alive by the Leak Suspect?

The Retained Set of the Accumulation Point
Agenda

- Automation for Technical **Support** Staff
  - Leak Suspects
  - Memory Intensive Threads
- Automation for **Developers**
- Questions & Answers
Demo
Problem Suspect 1

The thread org.mortbay.thread.BoundedThreadPool$PoolThread @ 0x200e108 btpool0-9 keeps variables with total size 64,746,336 (96.85%) bytes.

The memory is accumulated in one instance of "java.util.LinkedList$Entry" loaded by "<system class loader>".

The thread is executing an HTTP Request to /demo/index.jsp.

Keywords
java.util.LinkedList$Entry

Request URL
/demo/?flag=true&mbPerSecond=20 HTTP/1.1 Host: localhost:8088. Ur

More Request Details (Parameters etc.) provided by Pluggable Request Resolvers
What are we doing? (No Magic)

If the Suspect is a Thread, then

Extract Request Activity from <Java Local>s

Objects currently referenced from the call stack of the thread are marked <Java Local>
Agenda

• Automation for Technical **Support** Staff
  • Leak Suspects
  • Memory Intensive Threads

• **Automation for Developers**

• Questions & Answers
Developer Use Case - Goals

• Limit Analysis to Components, not the whole Heap
• Report on Memory Consumption & Leaks
  ❖ Excessive Use of Empty Collections
  ❖ Duplicate Strings
  ❖ Leaking OSGi Bundles
  ❖ …
• Drill-Down to the last Bits of the Object Graph (if necessary)
Demo

• Investigate Memory Consumption per Component
• “Leaking” Eclipse Bundles
• Finalizer
Degenerated Maps

- HashMap
- table
- key
- value
- next
- HashMap$Entry
Finalizer

• A Finalizer is a method that is executed when an object is garbage collected.

```
obj = new Object()
obj becomes unreachable
GC
obj added to finalization queue
obj finalized
GC
obj reclaimed
```

Finalizer in the Heap

### Finalizer In Processing

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io.InputStream @ 0x8d7299</td>
<td>16</td>
<td>40</td>
</tr>
</tbody>
</table>

### Finalizer Queue

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io.InputStream @ 0x8d7299</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d7298</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d7297</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d7296</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d7295</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d7294</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d67f8</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d67f7</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>java.io.InputStream @ 0x8d67f6</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>Total: 10 entries</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Finalizer Thread

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.ref.Finalizer$FinalizerThread @ 0x803320.Finalizer Native Stack, Thread</td>
<td>86</td>
<td>153</td>
</tr>
</tbody>
</table>

### Finalizer Thread Locals

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io.InputStream @ 0x8d7299</td>
<td>16</td>
<td>40</td>
</tr>
</tbody>
</table>

In Memory Analyzer run

Java Basics → Finalizer Overview

- Object currently processed
- Objects ready for Finalization in Processing Order
- The Finalizer Thread Object
- Object currently processed
Agenda

• Automation for Technical **Support** Staff
  - Leak Suspects
  - Memory Intensive Threads
• Automation for **Developers**
• **Questions & Answers**
Infos & Download

Eclipse

http://www.eclipse.org/mat/

Forum

eclipse.technology.memory-analyzer

(Old WIKI @ SAP)

• Download, Screen Cam, FAQ

Legal Notices

• Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. In the United States, other countries, or both.

• Other company, product, or service names may be trademarks or service marks of others.