JavaOne

Sun's 2004 Worldwide Java Developer Conference

java.sun.com/javaone/sf

INSANE

A simple way to diagnose memory leaks

Radim Kubacki

Petr Nejedly

Sun Microsystems, Inc.

Sun microsystems

Java

Sun

Presentation Goal

This session describes a technique for tracing heap memory from inside a running application. It will describe how to use this technique to locate memory leaks, and how to write automated memory consumption regression tests.

Agenda

- The technique
  - Modes of operation
  - Tracing
  - Finding roots
  - Limitations
- Usage
  - Heap evaluation
  - Regression testing
  - Post-problem analysis
- Other tools
- Conclusion

Agenda

- The technique
  - Modes of operation
  - Tracing
  - Finding roots
  - Limitations
- Usage
  - Heap evaluation
  - Regression testing
  - Post-problem analysis
- Other tools
- Conclusion

The technique

Overview

- Reflective walking of heap structures
- No JVM support required
- No application runtime overhead
- Has some limitations

The technique: Modes of operation

- Tracing single structure
  - Invoked programmatically e.g. from a test
  - To find out / verify its size
  - To check for unexpected outgoing references
- Dumping whole heap
  - Invoked by user
    - listen on a port, hidden shortcut, ...
  - Needs a root set
  - Can analyze the dump later with any number
  - of innovative queries

The technique: Example of the dump format

```
<insane>
...
<object id='1040' type='java.lang.String'
  size='40'/>
<ref to='1040'
  name='sun.misc.URLClassPath.USER_AGENT_
  JAVA_VERSION'/>
...
<object id='16fa' type='[C' size='48'
  value='ULA-Java-Version'/>
<ref from='1040' to='16fa'
  name='java.lang.String.value' />
...
</insane>
```

The technique: Tracing

- Single tracing engine
  - BFS
  - Uses reflection to find reference fields
  - Have to be careful to not explore itself
- Filtering
  - Stop at known outgoing references
  - Skip the tracing-support objects
  - Skip weak references
- Reporting
  - Reports found objects and references
  - Can be processed directly or stored in a dump

The technique: Engine

```
Queue.put(rootSet);

for ( Object obj : queue ) {
  if ( filter(obj) ) continue;

  for ( Class cls : classesOf(obj) ) {
    for ( Field fld : cls.getDeclaredFields() ) {
      Object inst = fld.get(obj);
      if ( visitObject(inst) ) queue.put(inst);
      visitReference(obj, inst, fld);
    }
  }
}
```

The technique: Finding roots

- Root types:
  - Static references
  - Native references
  - Stack references
- No JVM support
  - Only static references are reachable for engine!
  - Stack references are rare in an idle application
- Loaded classes list
  - User classes from ClassLoader
  - System classes found during tracking
  - java.lang.instrument.Instrumentation

The technique: Limitations

- Operating from inside of JVM
  - Not all root references available
  - Can't instrument itself
  - Can cause application code to be invoked
  - Operates in the same heap as the application
  - Nontrivial object identification
- Nonstandard tool

Agenda

- The technique
  - Modes of operation
  - Tracing
  - Finding roots
  - Limitations
- Usage
  - Heap evaluation
  - Regression testing
  - Post-problem analysis
- Other tools
- Conclusion

Usage:
Heap evaluation

- Parse the dump
- Comparing statistics
  - between several dumps during one run
  - throughout the development cycle
- Looking for interesting patterns
  - Empty maps
  - Strings with lot of redundant space
  - Redundant strings
- Attributing heap usage to subsystems

Usage: Regression testing

- Trace a heap subgraph directly and verify its size after some operation
  - NbTestCase.assertSize(msg, limit, object)
- Verify object is GCed, find references from roots if not
  - WeakReference, gc(), generate and process dump

Usage:
Post-problem analysis

- Leave a hook in a production application
- If it grows during usage, dump the heap
  - Auto-dump on OOME possible
- Perform queries on the dump
  - Statistics -> what to focus on
  - References from roots -> who holds onto an object
  - Known cache entry points -> cache limits
- Due to zero runtime overhead, you can check the heap even in production environment
- Not limited to memory leak analysis

Agenda

- The technique
  - Modes of operation
  - Tracing
  - Finding roots
  - Limitations
- Usage
  - Heap evaluation
  - Regression testing
  - Post-problem analysis
- Other tools
- Conclusion

Other tools

- Usual profilers
- hprof + HAT
- HeapRoots
- JFluid

Agenda

- The technique
  - Modes of operation
  - Tracing
  - Finding roots
  - Limitations
- Usage
  - Heap evaluation
  - Regression testing
  - Post-problem analysis
- Other tools
- Conclusion

Conclusion

- Use proper tool for the purpose
- Can analyze the heap even in production environment
- Can check any heap structure
- Can write memory unit tests

For More Information

- URLs:
  - http://performance.netbeans.org/
- The NetBeans CVS
  - :pserver:anoncvs@cvs.netbeans.org/cvs
  -

Q&A

JavaOne

Sun's 2004 Worldwide Java Developer Conference

java.sun.com/javaone/sf

INSANE

A simple way to diagnose memory leaks

Radim Kubacki

Petr Nejedly

Sun Microsystems, Inc.

Sun microsystems

Java

Sun