Memory Profiling (Focused on Java SE)

Sang Shin
Michèle Garoche
http://www.javapassion.com
"Learn with Passion!"



#### **Topics**

- Memory profiling tools
- Heap profiling strategies
- Memory leak profiling strategies
- Generation count
- When to use what tools?
- Identifying problem memory usage patterns

# Memory Profiling Tools

#### Free Profilers: NetBeans Profiler

- Memory profiling / heap profiling
- Memory leak detection
- Supported platforms; Solaris (SPARC & x86), Linux, Windows and Mac OS X
- Requires HotSpot JDK 5 or later
- Included out-of-the-box in NetBeans IDE 6.x

#### Free Profilers: jmap/jhat

- jmap produces snapshot heap profile
- jhat reads and presents the data
  - Shipped with JDK 5 and later
  - Command line tool
  - > Heap memory profiling
  - Perm gen statistics
  - > Finalizer statistics
  - > Supported on all platforms

## Heap Profiling Tips

#### **Heap Profiling**

- Heap profiling provides information about the memory allocation footprint of an application.
- When is heap profiling needed or beneficial?
  - > Observing frequent garbage collections
  - > Application requires a large Java heap
  - Can be useful for obtaining better cpu utilization or application throughout and responsiveness
    - Less time allocating objects and/or collecting them means more cpu time spent running the application.

#### **Heap Profiling Tips: Strategies**

- What approaches work best for heap profiling
  - Start with holistic approach to isolate major memory allocators.
    - Look at objects with large amount of bytes being allocated.
    - Look at objects with large number of object allocated.
    - Look at stack traces for locations where large amounts of bytes are being allocated.
    - Look at stack traces for locations where large number of objects are being allocated.

#### **Heap Profiling Tips: Strategies**

- If holistic approach is too intrusive, NetBeans Profiler can profile subsets of the application.
  - > Hypothesize on packages or classes which might have a large memory allocation footprint.
    - Look at objects with large amount of bytes being allocated.
    - Look at objects with large number of object allocated.
    - Look at stack traces for locations where large amounts of bytes are being allocated.
    - Look at stack traces for locations where large number of objects are being allocated.

#### **Heap Profiling Tips: Strategies**

- Cross reference cpu profiling with heap profiling
  - Look for objects which may have lengthy initialization times and allocate large amounts of memory. They are good candidates for caching.
- Look for alternative classes, objects and possibly caching approaches for large allocators.
- Consider profiling while application is running to observe memory allocation patterns.

#### **Profiling Tips: Strategies**

- jmap and jhat can also do heap profile
  - Not as sophisticated as NetBeans Profiler
  - Limited to a snapshot at the time of jmap capture. (jmap captures the snapshot, jhat displays the data)
  - User interface not as polished as NetBeans Profiler
  - Easily view top memory consumer at time when snapshot was taken.
  - Look at stack traces for allocation location.

#### **Profiling Tips: Strategies**

- Focus on large memory allocators
  - Consider alternative classes, objects and possibly caching approaches for large allocators.
- Capture several snapshots. Compare top memory allocators.

#### **Profiling Tips: Strategies**

- Quick and easy to use
  - > run jmap on the command line
  - run jhat on the command line
  - > connect with a web browser
- Can be intrusive on the application to generate the snapshot.

## Memory Leak Profiling Tips

- Memory leaks are situations where a reference to allocated object(s) remain unintentionally reachable and as a result cannot be garbage collected.
- Lead to poor application performance.
- Can lead to application failure.
- Can be hard to diagnose.

- Tools which help find memory leaks
  - NetBeans Profiler
  - > VisualVM
  - > jmap / jhat
  - Commercial offerings (not covered in this presentation)
    - JProbe Memory
    - YourKit
    - SAP Memory Analyzer

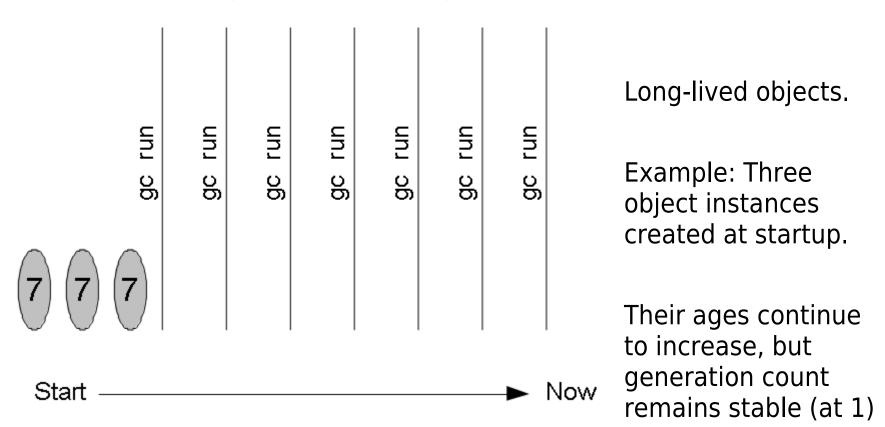
- NetBeans Profiler / VisualVM Strategies
  - View live heap profiling results while application is running.
  - > Pay close attention to "Surviving Generations".
    - Surviving Generations is the number of different object ages for a given class.
    - An increasing Surviving Generations over a period of time can be strong indicator of a source of a memory leak.

- jmap / jhat Strategies
  - Capture multiple heap profiles and compare footprints, (i.e. look for obvious memory usage increases).
  - -XX:+HeapDumpOnOutOfMemoryError
    - Use this JVM command line switch when launching application: heap dump will be created when OutOfMemory condition occurs.
    - Can be used with-XX:HeapDumpPath=<path>/<file>
  - Use jhat's OQL to query with interesting state information, (i.e live HTTP requests):
    - select s from com.sun.grizzly.ReadTask s s.byteBuffer.position > 0

## Generation Count

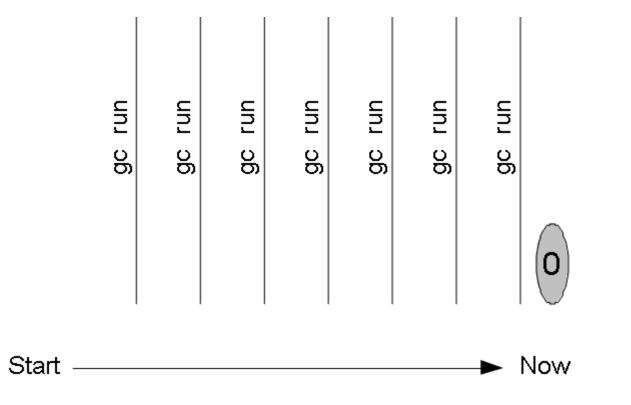
#### **How Does "Generation Count" Help?**

One Example of Healthy Behavior:



#### **How Does "Generation Count" Help?**

Another Example of Healthy Behavior:



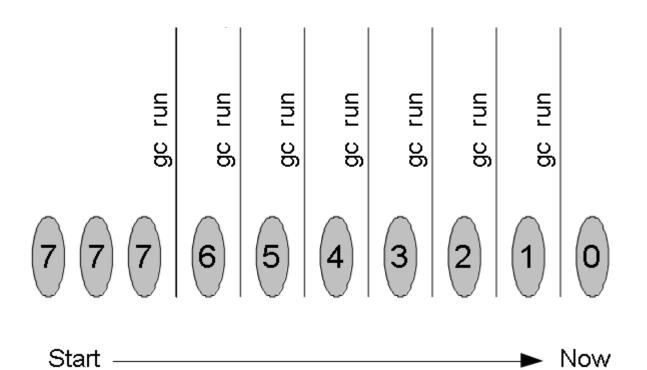
Short-lived objects

Example: Create an object, use it and then immediately let go of all references to it.

Generation count remains stable (at 1)

#### **How Does "Generation Count" Help?**

Unhealthy Behavior (A Memory Leak):



Example: Continue to allocate objects without letting go of all references.

Ten objects with eight different ages.

Generation count is **always increasing**.

## When To Use Which Tool?

- NetBeans Profiler
  - Profiling subset of application, for CPU profiling or heap profiling
  - > Heap profiling
  - > Finding memory leaks
  - Profiling an application using NetBeans IDE and/or NetBeans project
  - > Remote profiling
  - Attach to running application (with JDK 6)
  - > View profiling as application is running

- jmap / jhat
  - > Heap profiling
  - > Finding memory leaks
  - > Simple command line utilities
  - > Quick & easy snapshots of running application

- Sun Stuido Collector / Analyzer
  - Solaris and Linux platforms
  - > CPU profiling entire application
  - Sys cpu profiling or distinct usr vs sys profiling
  - Lock contention profiling
  - Integration with scripts, command files or batch files
  - Also view performance of JVM internals including methods
  - > Want to see machine level assembly instructions
  - Narrow to specific window of sampling

- DTrace and DTrace scripts
  - > Solaris platform
  - Non-intrusive snapshots of running application
  - Command line utility
  - Can leverage existing public scripts
    - Heap profiling
    - Finding memory leaks
    - Monitor contention
    - JIT Compilation
    - Garbage collection activity
    - Method entry / exit
    - Java Native Interface entry / exit

### Identifying Problem Memory Usage Patterns (Anti-Patterns)

#### Problem-patterns in heap profile

- Large number of String or char[] allocations in heap profile
  - Possible over allocation of String
  - Possibly benefit from use of StringBuilder
  - Possible StringBuilder or StringBuffer resizing.
  - Possibly utilize ThreadLocal to cache char[] or StringBuilder or StringBuffer
- Reducing char[] allocations will likely reduce garbage collection frequency.

- Observing StringBuffer in heap profile
  - Possible candidate for StringBuilder if synchronized access is not required.
  - > Reducing char[] allocations on StringBuilder/StringBuffer on expansion of StringBuilder/StringBuffer size.

- Observing Hashtable in heap profile
  - Possible candidate for HashMap if synchronized access is not required.
  - Possible candidate for ConcurrentHashMap if synchronized access is required.
  - Further partitioning of data stored in Hashtable may lead to finer grained synchronized access and less contention.
- Removing unnecessary synchronization will likely reduce sys cpu time spent spinning on locks.

- Observing Vector in heap profile
  - Possible candidate for ArrayList if synchronized access is not required.
  - If synchronized access required and depending on its use, consider using: LinkedBlockingDeque, ArrayBlockingQueue, ConcurrentLinkedQueue, LinkedBlockingQueue or PriorityBlockingQueue.
- Removing unnecessary synchronization will likely reduce sys cpu time spent spinning on locks.

- Exception object allocations
  - Possibly using exceptions for flow control
  - Use alternative flow control ... if / then / else, or switch flow control
- Generating stack traces are expensive operations.

#### Memory leaks in heap profiles

- Monitoring for trends illustrating increasing "surviving generations" while heap profiling when application is running indicates strong memory leak candidate.
  - > See section on Tools for Profiling.
- -XX:HeapDumpOnOutOfMemoryError can be used to capture heap dumps when out of memory errors occur.
  - Heap dumps can be analyzed by JHAT, NetBeans Profiler or VisualVM.

#### Thank you!



Check JavaPassion.com Codecamps! http://www.javapassion.com/codecamps