OutOfMemoryError: Quel est le coût des objets en Java

Jean-Philippe BEMPEL @jpbempel
Architecte Performance - Ullink
http://jpbempel.blogspot.com
jpbempel@ullink.com

Brown B G Lunch
P RIS

peRf
UserGroup
Paris

Agenda

- java.lang.Object
- CompressedOops
- Taille des structures
- Diagnostique
- Solutions

java.lang.Object

java.lang.Object

Fields	**32 bits**	**64 bits**	**64 bits** **CompressedOops**
mark word	4 bytes	8 bytes	8 bytes
klass pointer	4 bytes	8 bytes	4 bytes
Total	8 bytes	16 bytes	12 bytes (but 16 with padding/alignment)

java.lang.Object

Mark Word

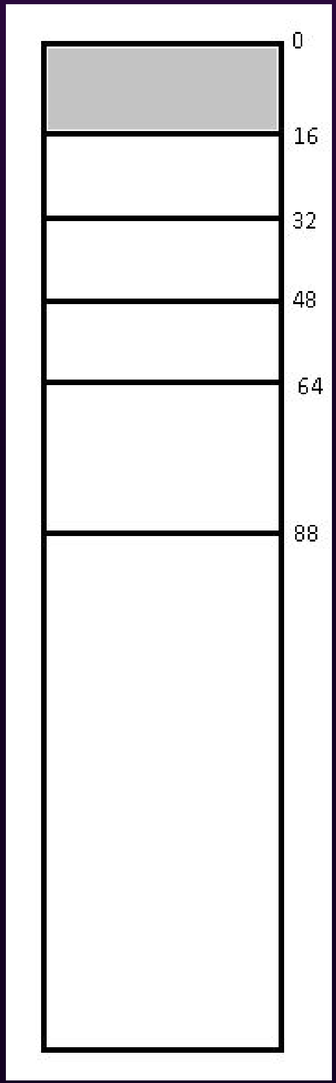
Bitfields			Tag	State
Hashcode	Age	0	01	Unlocked
Lock record address			00	Light-weight locked
Monitor address			10	Heavy-weight locked
Forwarding address, etc.			11	Marked for GC
Thread ID	Age	1	01	Biased / biasable

CompressedOops

CompressedOops

- Pointeur 32 bits ne peut adresser que 4 Go
- Adresse mémoire sont alignés (4 ou 8 octets)
- Objets résident seulement aux adresses multiple de 8
- 3 derniers bits ne sont pas utilisés
- Peut être utiliser pour augmenter l’espace adressable

CompressedOops



Example:

Adresse 88 en binaire:
101 1000
sera encodé par (décalage de 3 bits)
1011

Permet d'encodé 8 fois plus d'adresses qu'avec un encodage 32 bits classique
=> maximum adressable est maintenant de 32Go

Gain avec CompressedOops ~20-30% en mémoire

Activé par défaut sur JVM 64bits
JVM option: `-XX:+UseCompressedOops`

Padding

Padding

Avec un alignement de 8 octets, le padding intervient

Quel est la taille réelle de cette classe :

```
class Data
{
 long l;
 boolean b;
 int i;
 char c;
 String str;
}
```

Padding

L’analyse de heap dump avec un profiler peut donner une bonne estimation

La taille réelle dépend du mode : 32bits/64bits/CompressedOops

Pour être précis : Java Object Layout (JOL)

```
java -XX:+UseCompressedOops -jar jol-internals.jar java.lang.Object
Running 64-bit HotSpot VM.
Using compressed references with 3-bit shift.
Objects are 8 bytes aligned.
Field sizes by type: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
Array element sizes: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]

java.lang.Object object internals:
 OFFSET  SIZE  TYPE DESCRIPTION            VALUE
      0     4       (object header)        01 00 00 00 (0000 0001 0000 0000 0000 0000 0000 0000)
      4     4       (object header)        00 00 00 00 (0000 0000 0000 0000 0000 0000 0000 0000)
      8     4       (object header)        6d 05 88 df (0110 1101 0000 0101 1000 1000 1101 1111)
     12     4       (loss due to the next object alignment)
Instance size: 16 bytes (estimated, add this JAR via -javaagent: to get accurate result)
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```

Padding

Classe Data :

```
java -XX:+UseCompressedOops -classpath .;jol-internals.jar org.openjdk.jol.MainObjectInternals Data
Running 64-bit HotSpot VM.
Using compressed references with 3-bit shift.
Objects are 8 bytes aligned.
Field sizes by type: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
Array element sizes: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]

Data object internals:
 OFFSET  SIZE    TYPE DESCRIPTION          VALUE
      0     4         (object header)      01 00 00 00 (0000 0001 0000 0000 0000 0000 0000 0000)
      4     4         (object header)      00 00 00 00 (0000 0000 0000 0000 0000 0000 0000 0000)
      8     4         (object header)      b6 8c 91 df (1011 0110 1000 1100 1001 0001 1101 1111)
     12     4     int Data.i                           0
     16     8    long Data.l                           0
     24     2    char Data.c
     26     1 boolean Data.b                           false
     27     1         (alignment/padding gap)          N/A
     28     4  String Data.str                         null
Instance size: 32 bytes (estimated, add this JAR via -javaagent: to get accurate result)
Space losses: 1 bytes internal + 0 bytes external = 1 bytes total
```

Taille des structures

Arrays

Arrays ont un champ additionnel pour la taille (int)

En-têtes (64bits) + CompressedOops: 16 octets

```
byte[32]   => header + 1*32 bytes = 48 bytes
short[32]  => header + 2*32 bytes = 80 bytes
char[32]   => header + 2*32 bytes = 80 bytes
int[32]    => header + 4*32 bytes = 144 bytes
long[32]   => header + 8*32 bytes = 272 bytes
double[32] => header + 8*32 bytes = 272 bytes
Object[32] => header + RefSize*32 bytes = 144 bytes
```

java.lang.String

1.6.0_45 (32 bytes)
- char[] value
- int hash
- int count
- int offset

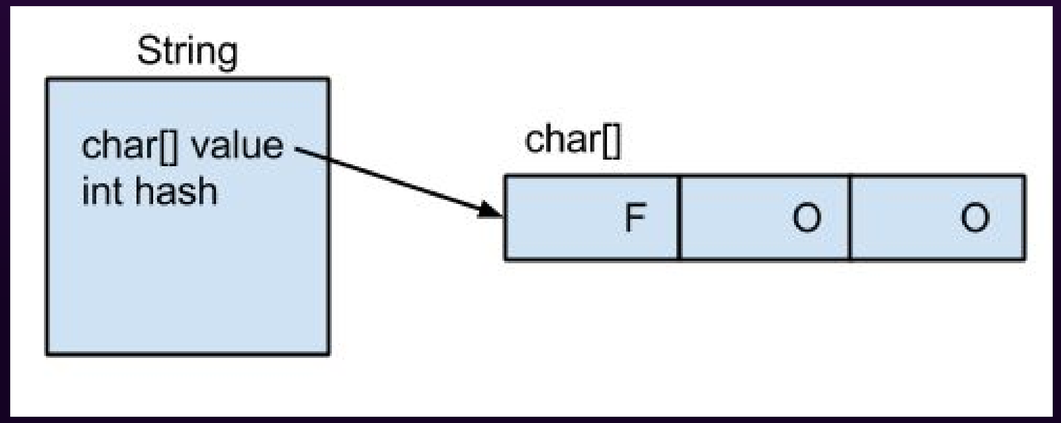
1.7.0_55 (24 bytes)
- char[] value
- int hash
- int hash32

1.8.0_60 (24 bytes)
- char[] value
- int hash

String.subString()
- <1.7.0_06 => shared char[]
- >= 1.7.0_06 => make copy of char[]

java.lang.String

Class String + char[]



Exemple pour la string `foo`

`Class String Size + char array header + 2*3 bytes = 24 + 16 + 6 = 46 bytes`

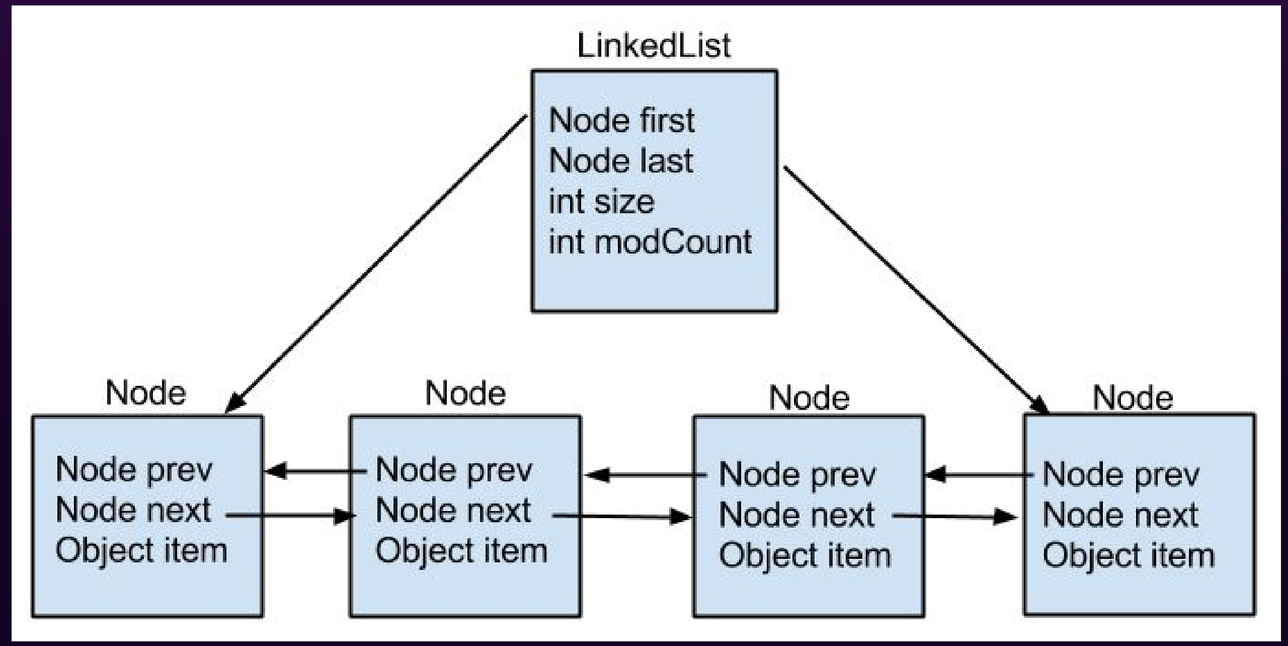
Surcoût = 1433%

Pour string de 100 caractères :

`Class String Size + char array header + 2*100 bytes = 24 + 16 + 200 = 240 bytes`

Surcoût = 140%

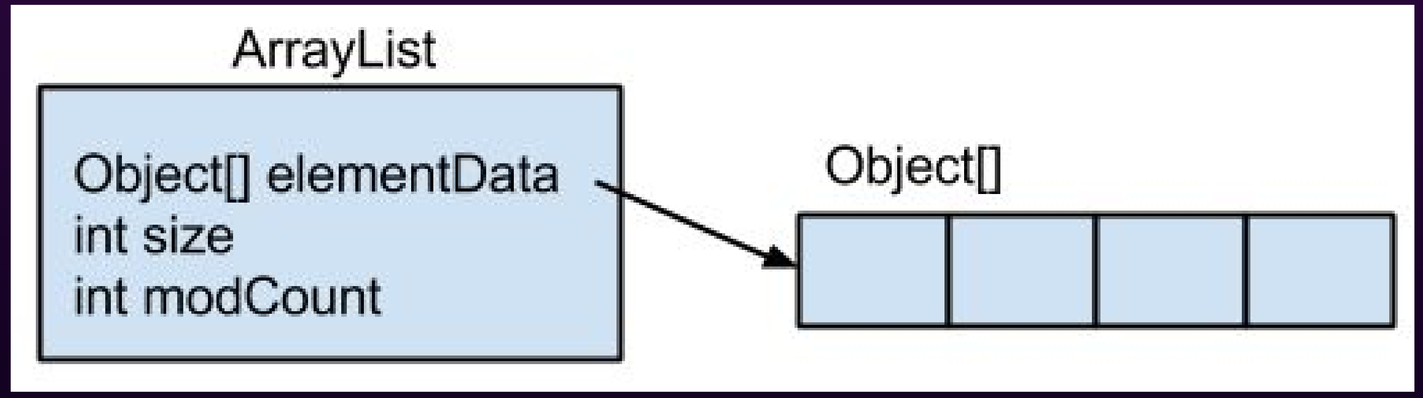
java.util.LinkedList



Classe LinkedList : 32 octets
Classe Node : 24 octets

Exemple pour 100 éléments : 32 + 100*24 = 2432 octets

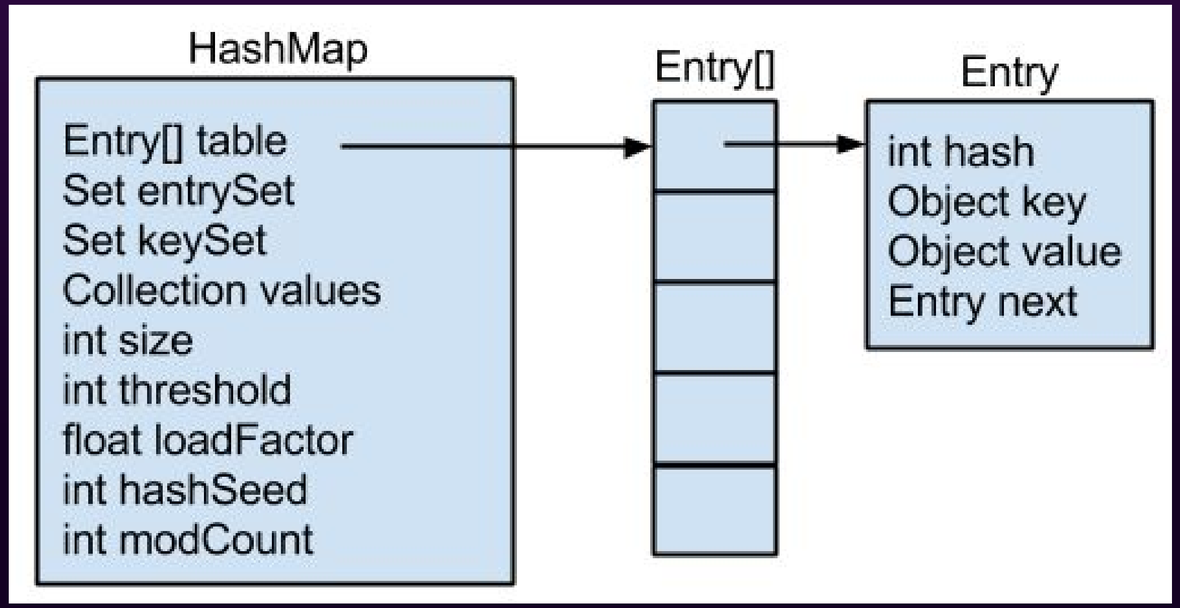
java.util.ArrayList



Classe ArrayList : 24 octets
Object[] : 16 + n*4 octets

Exemple pour 100 éléments : 24 + 16 + 100*4 = 440 octets

java.util.HashMap



Classe HashMap : 48 octets
Entry[] : 16 + n*4 octets
Classe Entry : 32 octets

Exemple pour 100 paires clé/valeurs :
48 + 16 + 256*4 + 100*32 = 4288 octets

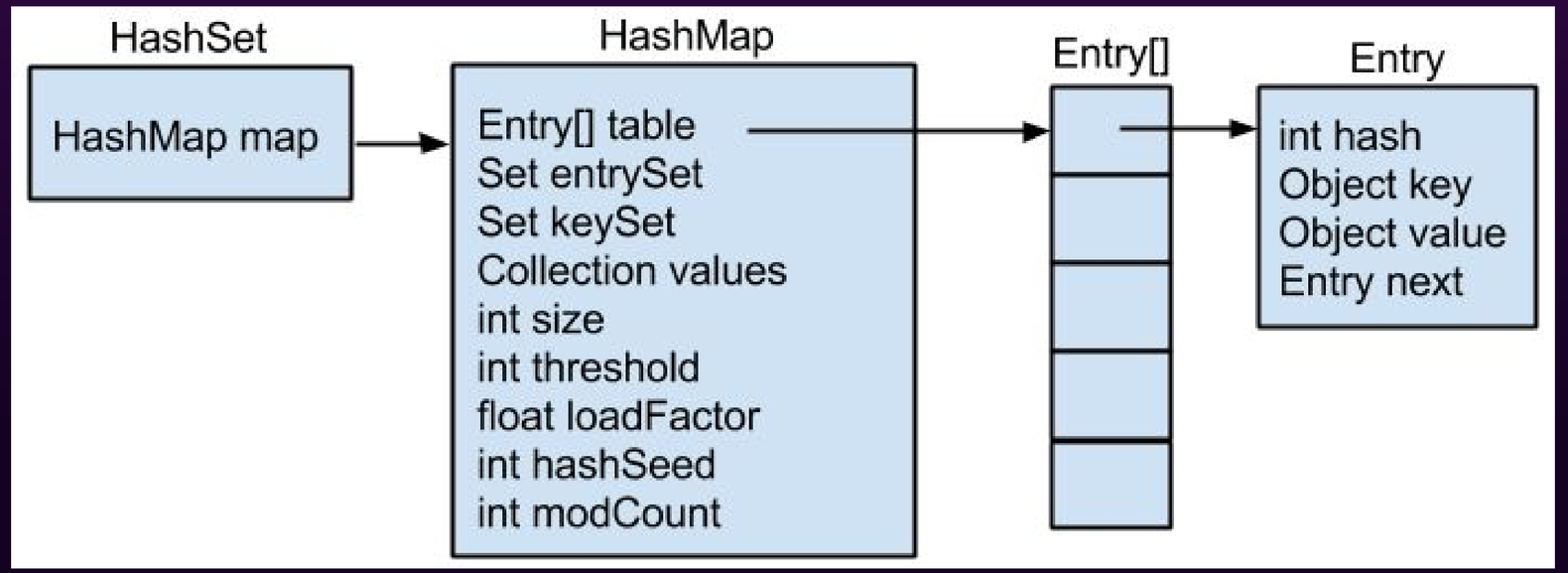
java.util.HashMap

- entrySet() appelée => +1 instance EntrySet (16 octets)
- keySet() appelée => +1 instance KeySet (16 octets)
- values() appelée => +1 instance Values (16 octets)

Pour ces inner classes, on a l'en-tête Object + la référence de la classe "outer"

```
java.util.HashMap.Values object internals:
 OFFSET  SIZE    TYPE DESCRIPTION                    VALUE
      0    12         (object header)                N/A
     12     4 HashMap Values.this$0                  N/A
Instance size: 16 bytes (estimated, the sample instance is not available)
Space losses: 0 bytes internal + 0 bytes external = 0 bytes total
```

java.util.HashSet

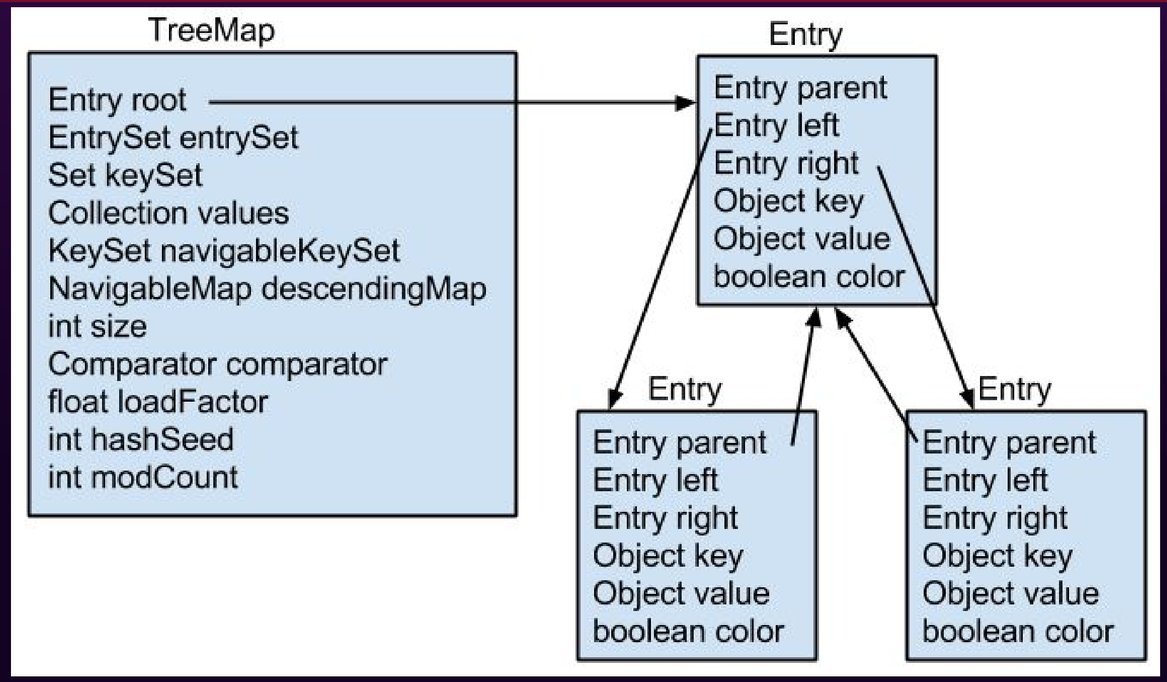


Classe HashSet : 16 octets

Exemple pour 100 éléments :

16 + coût d'une HashMap = 4304 octets

java.util.TreeMap



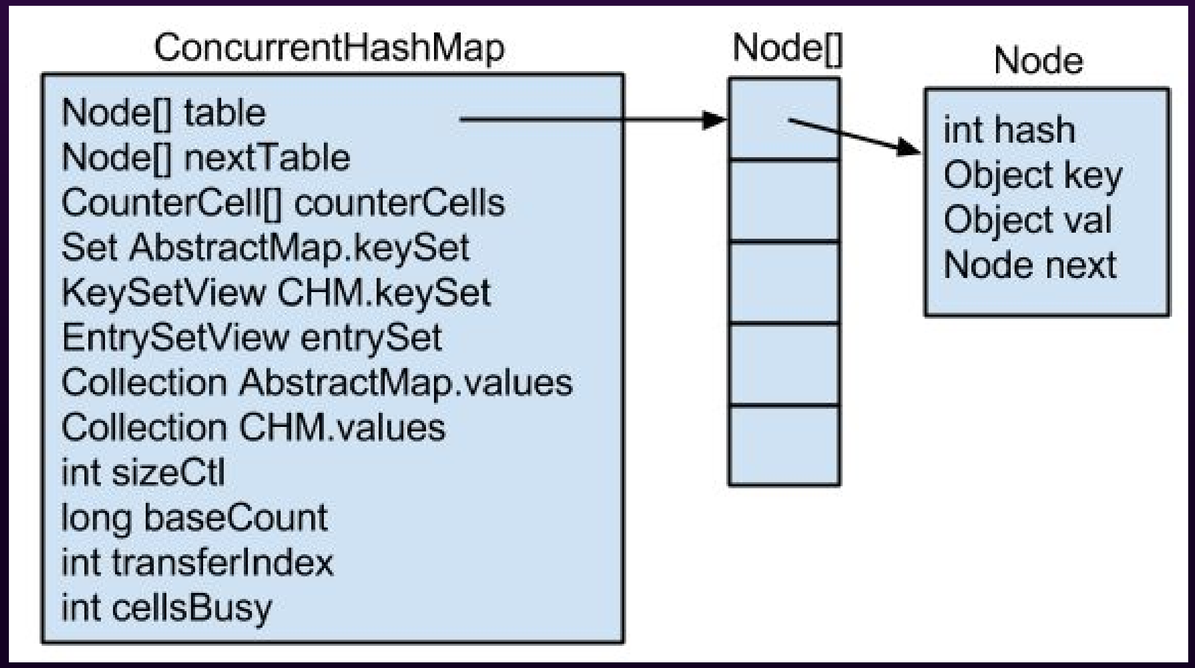
Classe TreeMap : 48 octets
Classe Entry : 40 octets (double pénalité)

Exemple pour 100 éléments :
48 + 100*40 = 4048 octets

java.util.TreeMap

- entrySet() appelée => +1 instance EntrySet (16 octets)
- keySet() appelée => +1 instance KeySet (16 octets)
- values() appelée => +1 instance Values (16 octets)
- navigableKeySet() appelée => +1 instance KeySet (16 octets)
- descendingMap() appelée => +1 instance DescendingSubMap (56 octets)

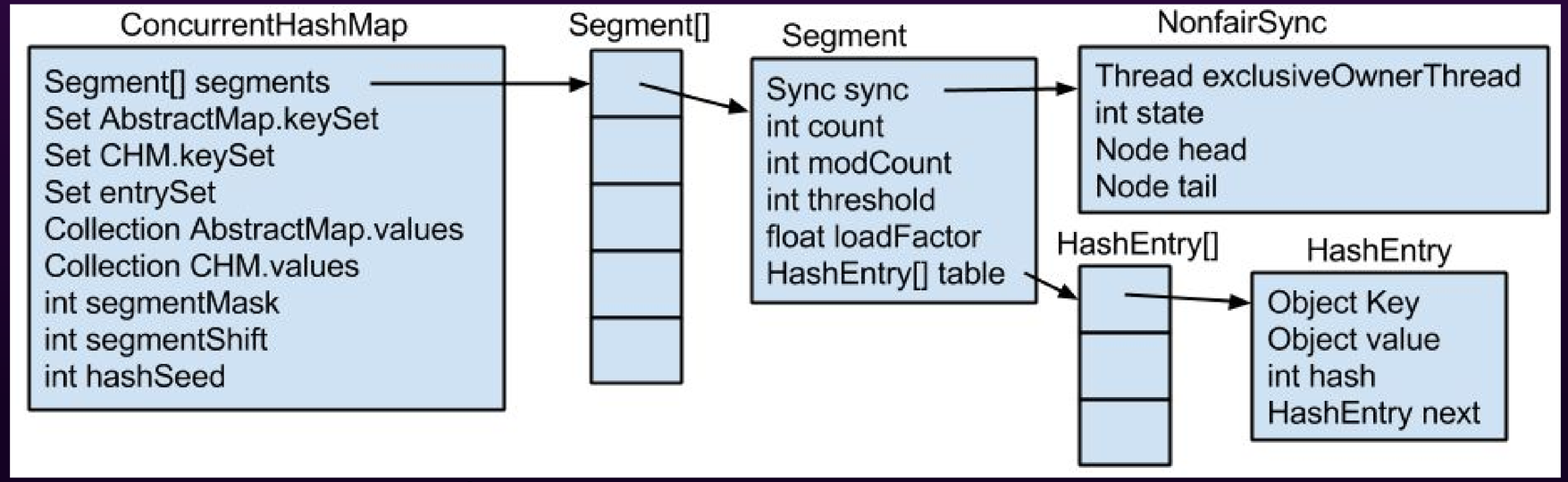
ConcurrentHashMap
(1.8)



Classe ConcurrentHashMap : 64 octets
Classe Node : 32 octets

Exemple pour 100 paires clé/valeurs :
64 + 16 + 256*4 + 100*32 = 4304 octets

ConcurrentHashMap
(1.7)



Classe ConcurrentHashMap : 48 octets
Segment[] : 16 + 16*4 = 80 bytes
Classe Segment : 40 octets + classe Sync : 32 octets
HashEntry[] : 16 + n*4 octets
Classe HashEntry : 32 octets

Exemple pour 100 paires clé/valeurs :
48 + 80 + 16*(40 + 32 +16 + 16*4) + 32 * 100= 5760 octets

Taille des structures

Collections	Surcoût pour 100
ArrayList	440 octets
LinkedList	2432 octets
TreeMap	4048 octets
HashMap	4288 octets
HashSet	4304 octets
ConcurrentHashMap (1.8)	4304 octets
ConcurrentHashMap (1.7)	5760 octets

Diagnostique

Histogramme de Classes

jmap -histo <pid>

```
 num     #instances         #bytes  class name
----------------------------------------------
   1:         73903        5978960  [C
   2:          4309        4085624  [I
   3:          2790        1730968  [B
   4:         49641        1191384  java.lang.String
   5:         22080         706560  java.util.HashMap$Node
   6:          5420         571328  java.lang.Class
   7:          7408         431080  [Ljava.lang.Object;
   8:          2908         331584  [Ljava.util.HashMap$Node;
   9:          1339         289224  sun.java2d.SunGraphics2D
  10:          2882         253616  java.lang.reflect.Method
  11:          6586         227248  [Ljava.lang.Class;
  12:          1632         223768  [Ljava.lang.String;
  13:          4973         198920  java.security.AccessControlContext
  14:          3880         186240  java.util.HashMap
  15:          4890         156480  java.util.Hashtable$Entry
  16:          5376         129024  java.lang.StringBuilder
  17:          3293         105376  java.lang.ref.WeakReference
  18:          1424         102528  java.awt.geom.AffineTransform
  19:          3186         101952  java.awt.Rectangle
  20:          3005          96160  java.util.concurrent.ConcurrentHashMap$Node
```

Option JVM: -XX:+PrintClassHistogramBeforeFullGC
Inclut l'histogramme de classes dans les logs GC

Heap Dump

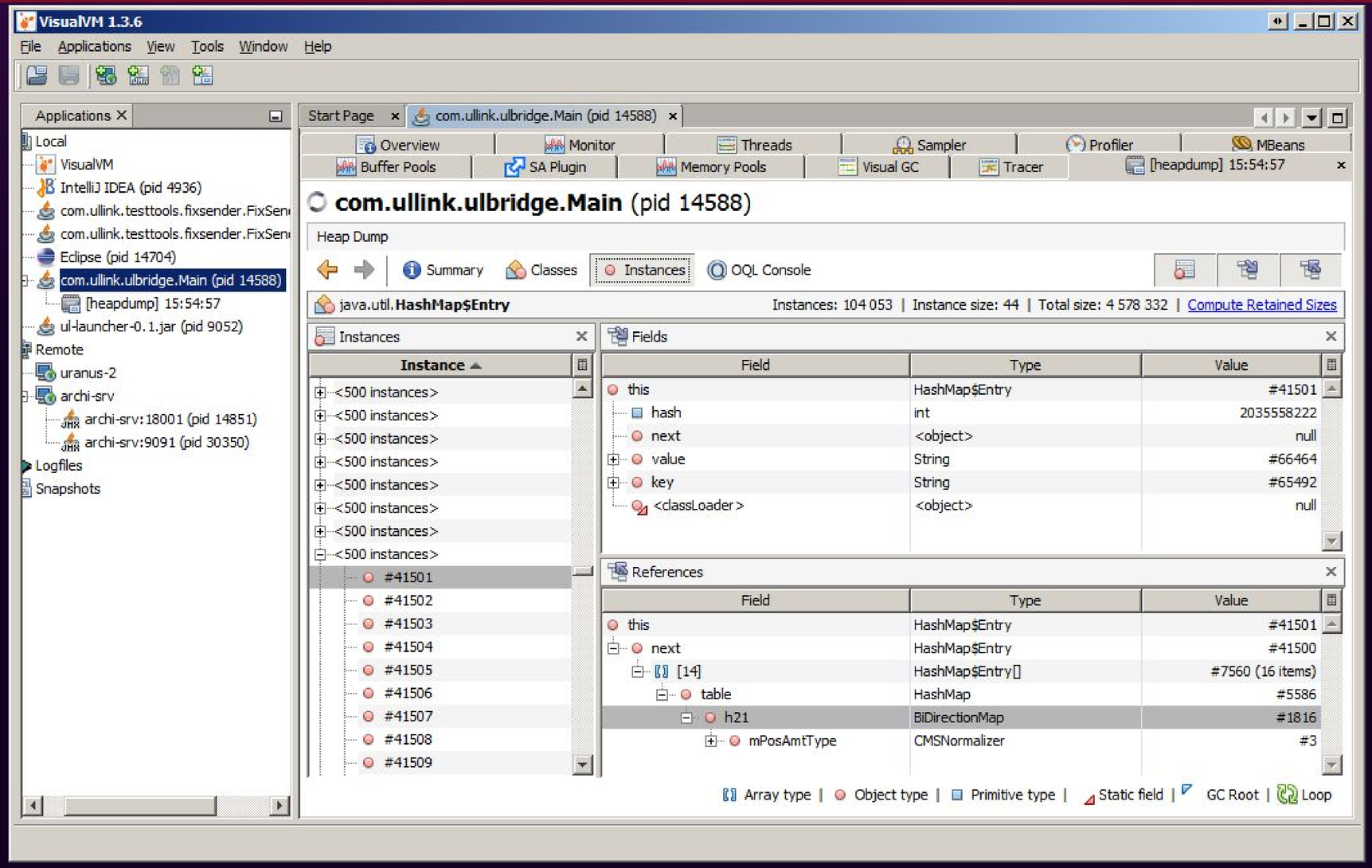
To generate it:

- jmap -dump:live,format=b,file=heap.hprof <pid>
- JVM Options:
  - -XX:+HeapDumpOnOutOfMemoryError
  - -XX:HeapDumpPath=../logs
- JMX: com.sun.management:type=HotSpotDiagnostic.dumpHeap(filename, liveonly)

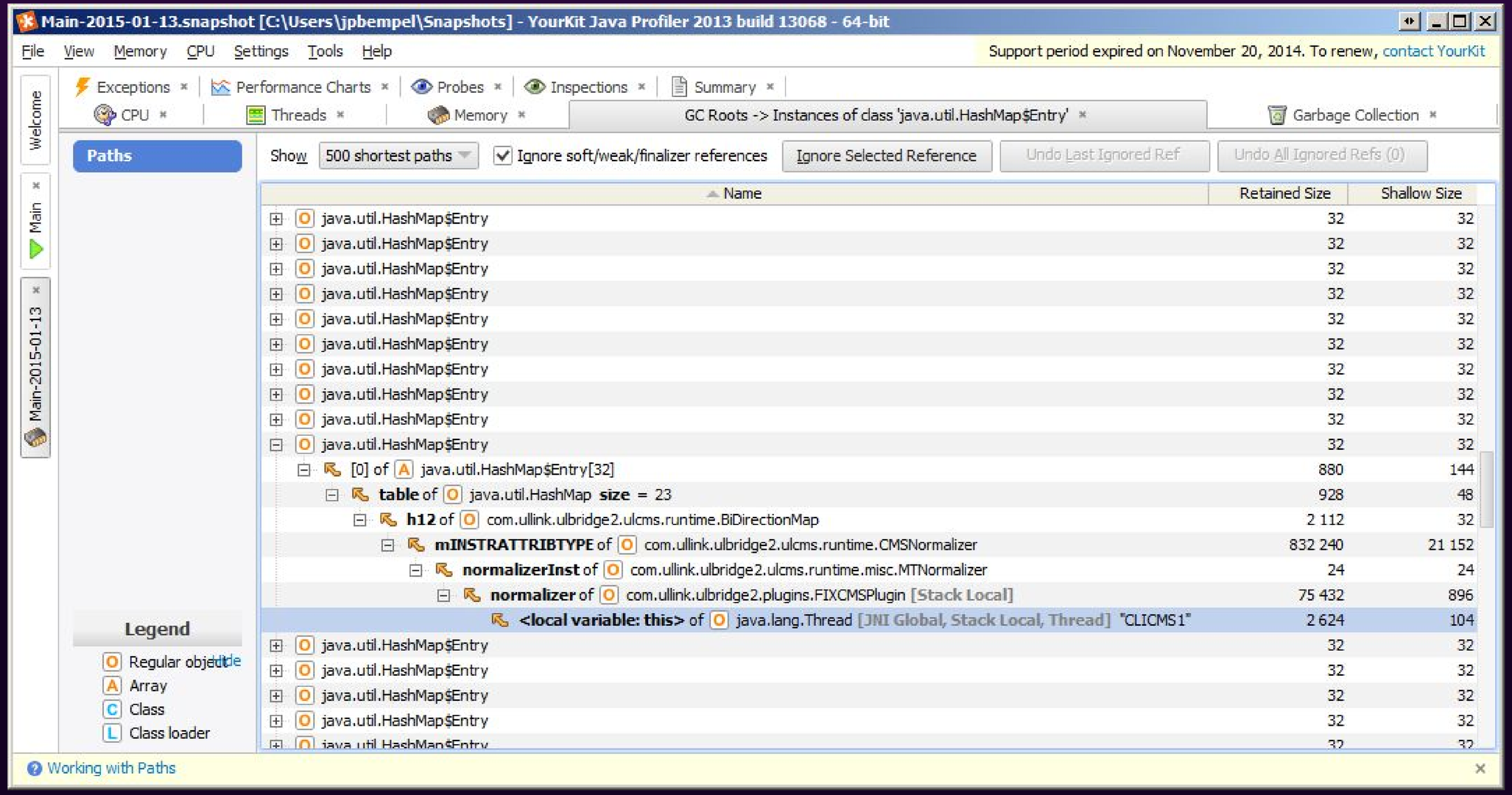
To exploit heap dump:

- visualVM
- YourKit
- Eclipse MAT

Heap Dump


VisualVM 1.3.6
File Applications View Tools Window Help
Applications
Local
VisualVM
IntelliJ IDEA (pid 4936)
com.ullink.testtools.fixsender.FixSen
com.ullink.testtools.fixsender.FixSen
Eclipse (pid 14704)
com.ullink.ulbridge.Main (pid 14588)
[heapdump] 15:54:57
ul-launcher-0.1.jar (pid 9052)
Remote
uranus-2
archi-srv
archi-srv:18001 (pid 14851)
archi-srv:9091 (pid 30350)
Logfiles
Snapshots
Start Page
com.ullink.ulbridge.Main (pid 14588)
Overview
Monitor
Threads
Sampler
Profiler
MBeans
Buffer Pools
SA Plugin
Memory Pools
Visual GC
Tracer
[heapdump] 15:54:57
com.ullink.ulbridge.Main (pid 14588)
Heap Dump
Summary
Classes
Instances
OQL Console
java.util.HashMap$Entry
Instances: 104 053 | Instance size: 44 | Total size: 4 578 332 | Compute Retained Sizes
Instances
Instance
<500 instances>
#41501
#41502
#41503
#41504
#41505
#41506
#41507
#41508
#41509
Fields
Field
Type
Value
this
HashMap$Entry
#41501
hash
int
2035558222
next
<object>
null
value
String
#66464
key
String
#65492
<classLoader>
<object>
null
References
Field
Type
Value
this
HashMap$Entry
#41501
next
HashMap$Entry
#41500
[14]
HashMap$Entry[]
#7560 (16 items)
table
HashMap
#5586
h21
BiDirectionMap
#1816
mPosAmtType
CMSNormalizer
#3
Array type | Object type | Primitive type | Static field | GC Root | Loop

Heap Dump


Main-2015-01-13.snapshot [C:\Users\jpbempel\Snapshots] - YourKit Java Profiler 2013 build 13068 - 64-bit
File View Memory CPU Settings Tools Help
Support period expired on November 20, 2014. To renew, contact YourKit
Exceptions Performance Charts Probes Inspections Summary
CPU Threads Memory GC Roots -> Instances of class 'java.util.HashMap$Entry' Garbage Collection
Welcome
Main
Main-2015-01-13
Paths
Show 500 shortest paths
Ignore soft/weak/finalizer references
Ignore Selected Reference
Undo Last Ignored Ref
Undo All Ignored Refs (0)
Name
Retained Size
Shallow Size
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
[0] of java.util.HashMap$Entry[32] 880 144
table of java.util.HashMap size = 23 928 48
h12 of com.ullink.ulbridge2.ulcms.runtime.BiDirectionMap 2 112 32
mINSTRATTRIBTYPE of com.ullink.ulbridge2.ulcms.runtime.CMSNormalizer 832 240 21 152
normalizerInst of com.ullink.ulbridge2.ulcms.runtime.misc.MTNormalizer 24 24
normalizer of com.ullink.ulbridge2.plugins.FIXCMSPlugin [Stack Local] 75 432 896
<local variable: this> of java.lang.Thread [JNI Global, Stack Local, Thread] "CLICMS1" 2 624 104
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
java.util.HashMap$Entry 32 32
Legend
Regular object
Array
Class
Class loader
Working with Paths

Solutions

Solutions

- Flyweight/internalizers
- ArrayList vs LinkedList
- HashMap => OpenAddressingHashMap

Measure, don’t guess!

Jack Shirazi & Kirk Pepperdine

Measure, don’t premature!

References

- Java Object Layout: http://openjdk.java.net/projects/code-tools/jol/
- What Heap Dumps Are Lying To You About: http://shipilev.net/blog/2014/heapdump-is-a-lie/
- From Java code to Java heap: http://www.ibm.com/developerworks/library/j-codetoheap/
- Building Memory-efficient Java Applications: Practices and Challenges: http://www.cs.virginia.edu/kim/publicity/pldi09tutorials/memory-efficient-java-tutorial.pdf

Merci

Jean-Philippe BEMPEL @jpbempel
Architecte Performance - Ullink
http://jpbempel.blogspot.com
jpbempel@ullink.com