



OutOfMemoryError: What is the cost of Java Objects?

Jean-Philippe BEMPEL
Performance Architect

@jpbempel
<http://jpbempel.blogspot.com>



Brown B G Lunch
P  RIS

Agenda

- `java.lang.Object`
- CompressedOops
- Structure sizes
- Troubleshooting
- War stories
- 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

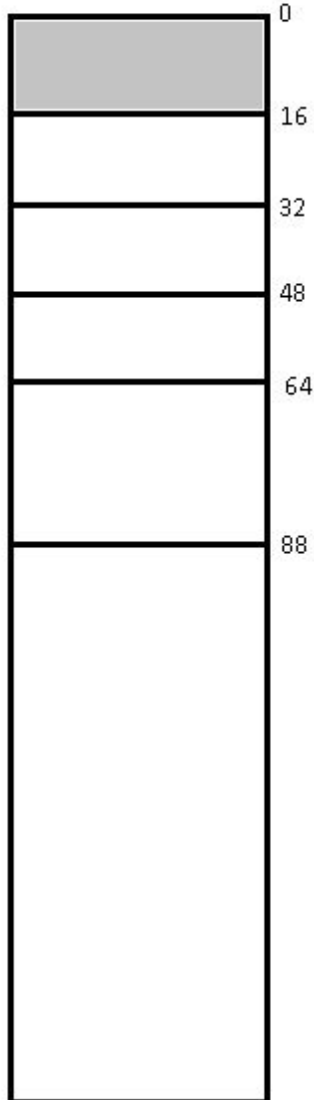
Bitfields			Tag	State
HashCode	Age	0	01	Unlocked
Lock record address			00	Light-weight locked
Monitor address			10	Heavy-weight locked
Forwarding address, etc.			11	Marked for GC
Thread ID	Age	1	01	Biased / biasable

CompressedOops

CompressedOops

- 32 bits pointer can address 4GB
- Memory addresses are aligned (4 or 8)
- Objects reside only on address 8 multiple
- Last 3 bits are not effectively used
- Use the 3 bits to increase addressable space

CompressedOops



Example:

Address 88 in binary:
101 1000
will be encoded as (3 bits shift)
1011

Allows to encode 8 times more object addresses then with raw 32 bits addressing
=> Maximum addressable is then 32GB

CompressedOops saves ~20-30% memory

Activated by default on 64bits JVM

JVM option: `-XX:+UseCompressedOops`

Padding

Padding

With 8 bytes field alignment, padding occurs

What is the real size of this class:

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

Padding

Heap Dump analysis with profilers can give a good estimation

Real size depends on 32bits/64Bits/CompressedOops

A precise way: 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

Class 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

Structure sizes

Arrays

Arrays have additional int for size

Header for 64 bits + CompressedOops: 16 bytes

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

java.lang.String

String class fields

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)

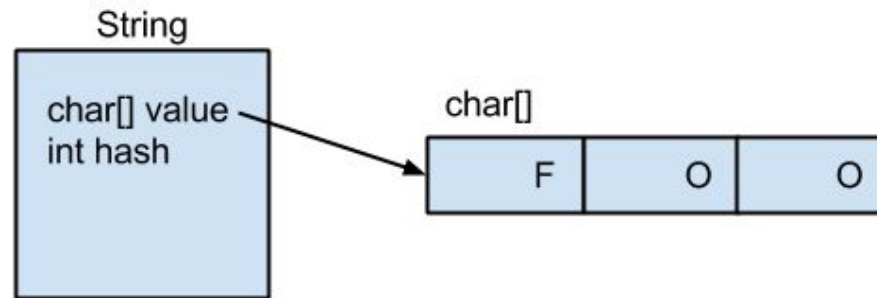
- char[] value
- int hash

String.substring()

- <1.7.0_06 => shared char[]
- >= 1.7.0_06 => make copy of char[]

java.lang.String

Class String + char[]



example for string f₀₀

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

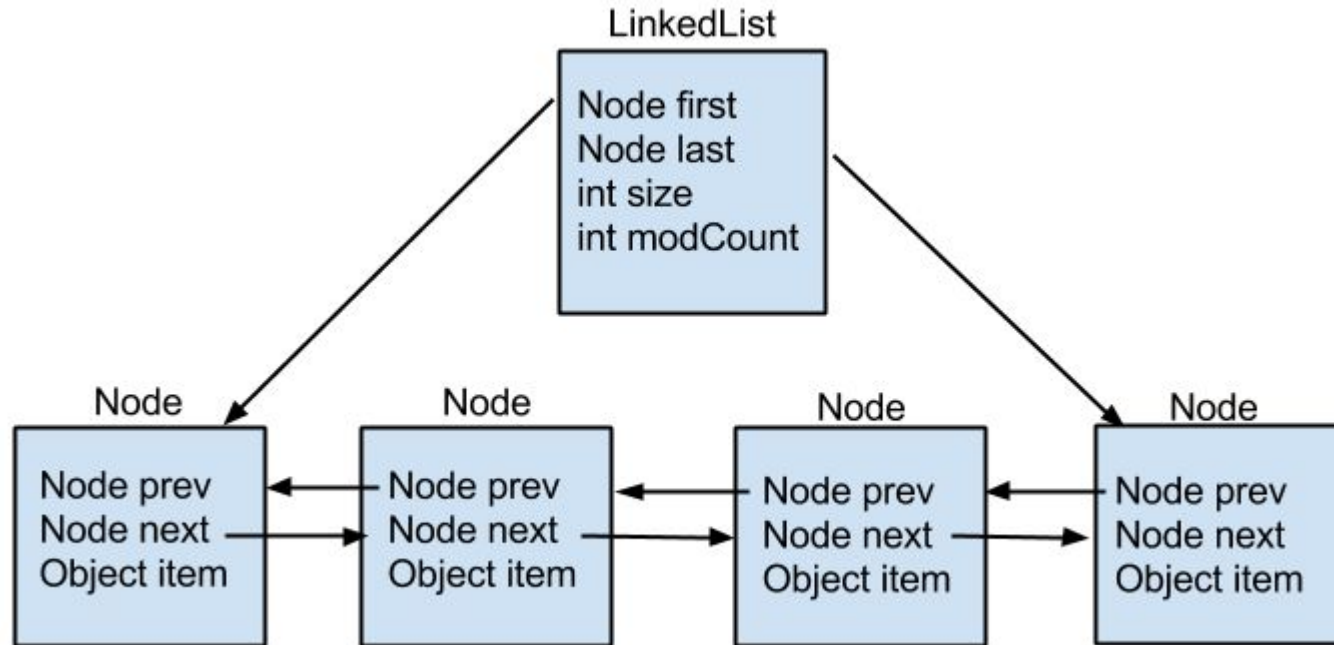
overhead = 1433%

for string of 100 characters:

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

overhead = 140%

java.util.LinkedList

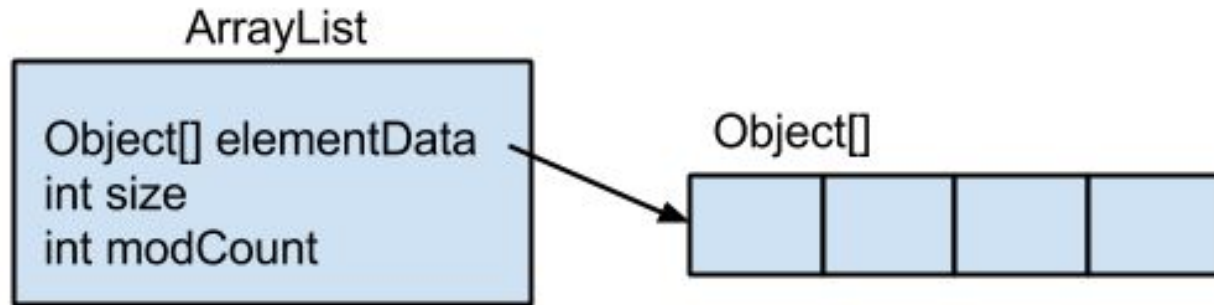


Class `LinkedList`: 32 bytes

Class `Node`: 24 bytes

Example for 100 elements: $32 + 100 \cdot 24 = 2432$ bytes

java.util.ArrayList

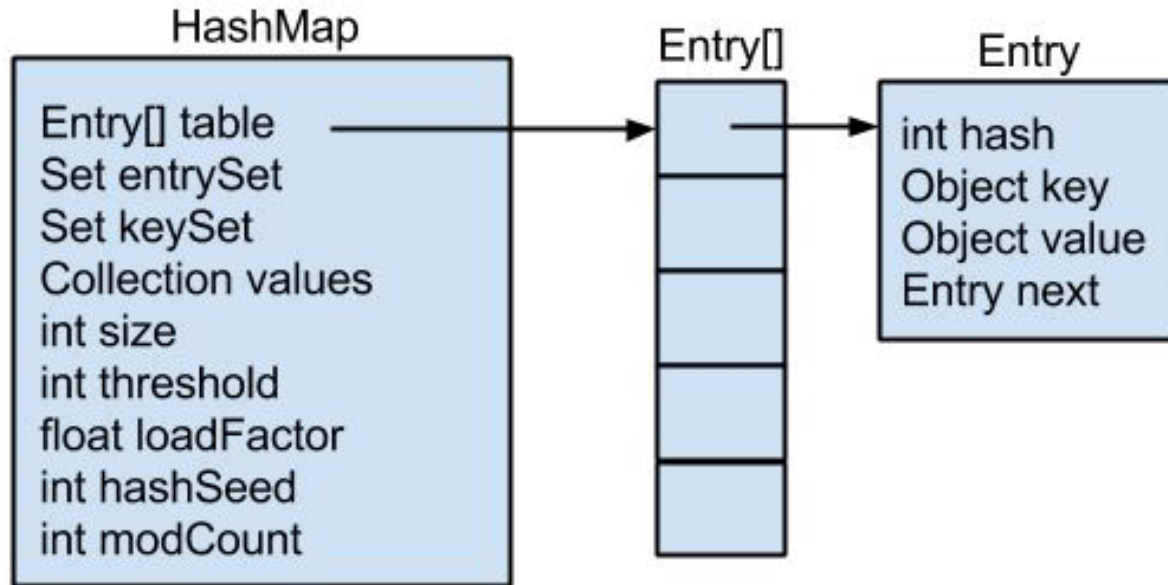


Class `ArrayList`: 24 bytes

`Object[]`: $16 + n \cdot 4$ bytes

Example for 100 elements: $24 + 16 + 100 \cdot 4 = 440$ bytes

java.util.HashMap



class HashMap: 48 bytes

Entry[]: $16 + n \cdot 4$ bytes

class Entry: 32 bytes

Example for 100 key/value pairs:

$$48 + 16 + 256 \cdot 4 + 100 \cdot 32 = 4288 \text{ bytes}$$

java.util.HashMap

- `entrySet()` called => add `EntrySet` instance (16 bytes)
- `keySet` called => add `KeySet` instance (16 bytes)
- `values()` called => add `Values` instance (16 bytes)

For those inner classes this is object header + outer ref

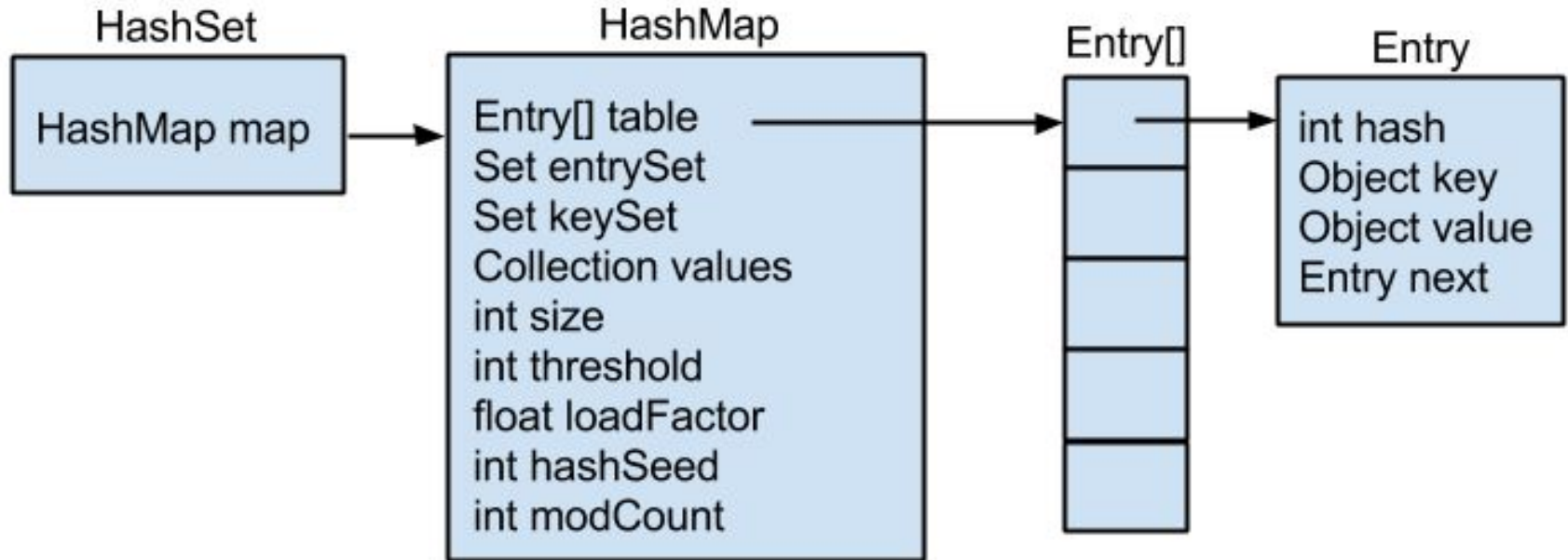
java.util.HashMap.Values object internals:

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

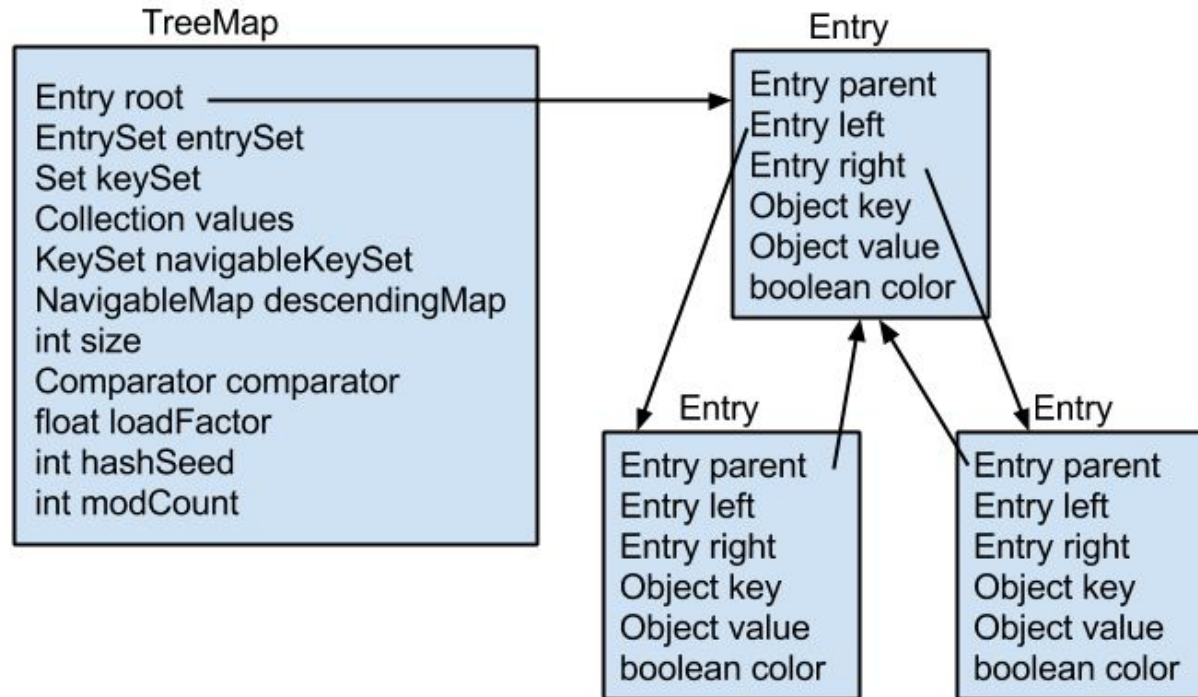
java.util.HashSet



class HashSet: 16 bytes

example for 100 elements: 16 + HashMap cost = 4304 bytes

java.util.TreeMap



class TreeMap: 48 bytes

class Entry: 40 bytes (double penalty)

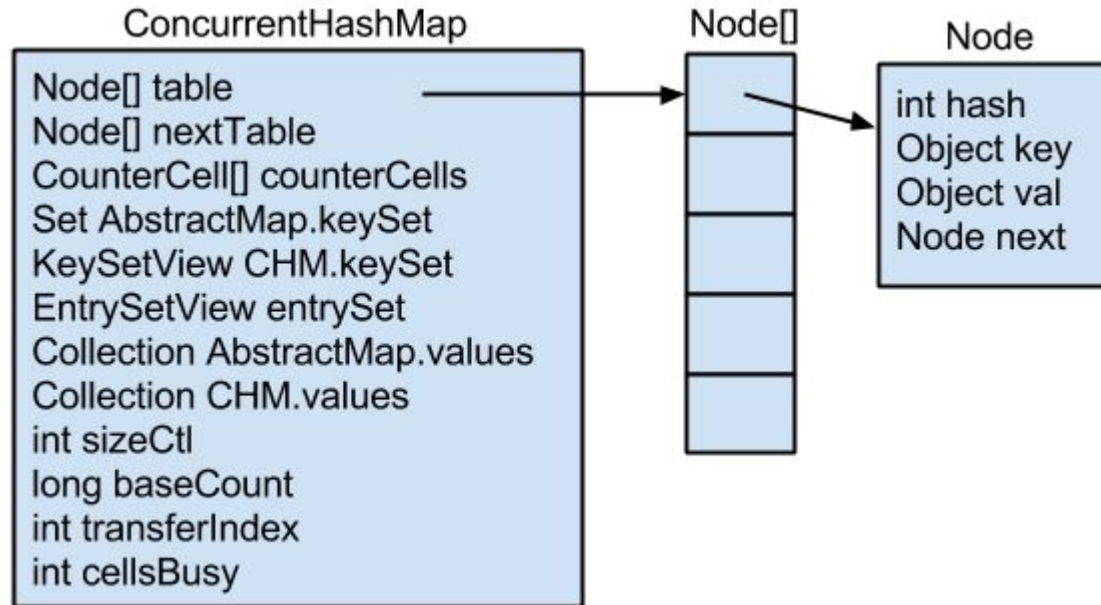
example for 100 elements:

$48 + 100 \cdot 40 = 4048$ bytes

java.util.TreeMap

- `entrySet()` called => add `EntrySet` instance (16 bytes)
- `keySet` called => add `KeySet` instance (16 bytes)
- `values()` called => add `Values` instance (16 bytes)
- `navigableKeySet` called => add `KeySet` instance (16 bytes)
- `descendingMap` called => add `DescendingSubMap` instance (56 bytes)

java.util.concurrent.ConcurrentHashMap (1.8)



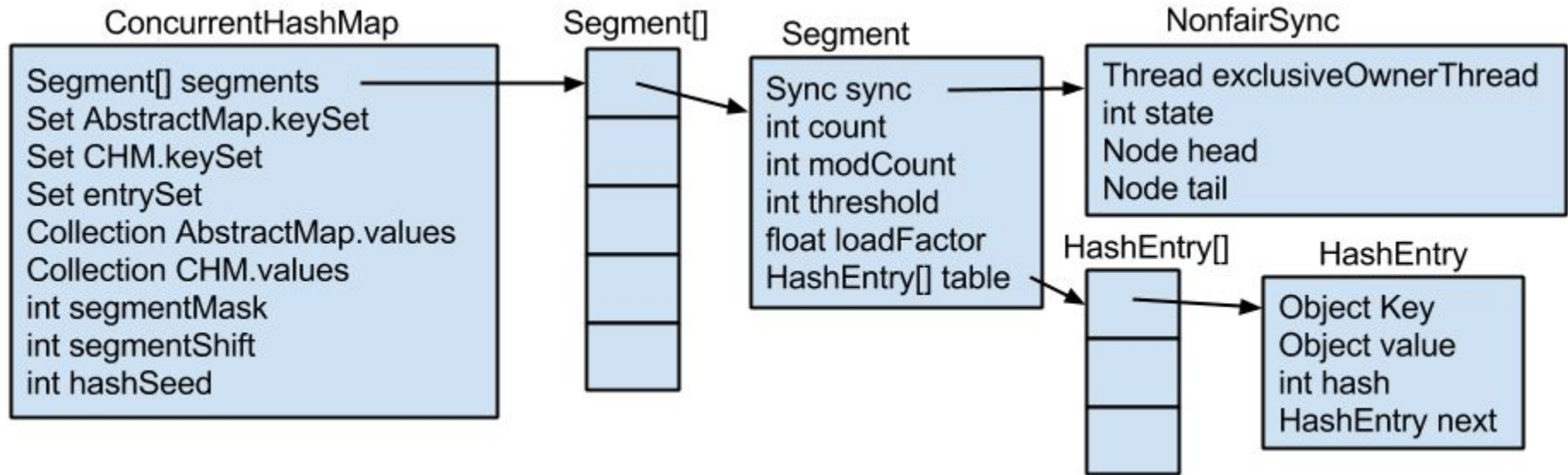
class `ConcurrentHashMap`: 64 bytes

class `Node`: 32 bytes

Example for 100 elements:

$$64 + 16 + 256 \cdot 4 + 100 \cdot 32 = 4304 \text{ bytes}$$

java.util.concurrent.ConcurrentHashMap (1.7)



class `ConcurrentHashMap`: 48 bytes

`Segment[]`: $16 + 16 \times 4 = 80$ bytes

class `Segment`: 40 bytes + class `Sync`: 32 bytes

`HashEntry[]`: $16 + n \times 4$ bytes

class `HashEntry`: 32 bytes

example for 100 elements:

$$48 + 80 + 16 \times (40 + 32 + 16 + 16 \times 4) + 32 \times 100 = 5760 \text{ bytes}$$

Summary

Collections	Overhead for 100
ArrayList	440 bytes
LinkedList	2432 bytes
TreeMap	4048 bytes
HashMap	4288 bytes
HashSet	4304 bytes
ConcurrentHashMap (1.8)	4304 bytes
ConcurrentHashMap (1.7)	5760 bytes

Troubleshooting

Troubleshooting: Class histogram

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

JVM Option: -XX:+PrintClassHistogramBeforeFullGC
Includes class histogram in GC logs

Troubleshooting: 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

Troubleshooting: Heap dump

VisualVM 1.3.6

File Applications View Tools Window Help

Applications X

Local

- VisualVM
- IntelliJ IDEA (pid 4936)
- com.ullink.testtools.fixsender.FixSender (pid 14588)
- 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 x com.ullink.ulbridge.Main (pid 14588) x

Overview Monitor Threads Sampler Profiler MBeans

Buffer Pools SA Plugin Memory Pools Visual GC Tracer [heapdump] 15:54:57 x

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 x Fields x

Instance	Field	Type	Value
<500 instances>	this	HashMap\$Entry	#41501
<500 instances>	hash	int	2035558222
<500 instances>	next	<object>	null
<500 instances>	value	String	#66464
<500 instances>	key	String	#65492
<500 instances>	<classLoader>	<object>	null

References x

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

Troubleshooting: 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 *

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
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
java.util.HashMap\$Entry	32	32
java.util.HashMap\$Entry	32	32

Legend

- Regular object
- Array
- Class
- Class loader

Working with Paths

War stories

War stories

Overhead structures example:

251614.001: [GC 251614.001: [ParNew (promotion failed): 943744K->940573K(943744K), 0.9248570 secs]251614.926:

[Class Histogram:

num	#instances	#bytes	class name

1:	421751083	20244051984	java.util.concurrent.ConcurrentHashMap\$HashEntry
2:	23327688	12130397760	com.ullink.ulmonitoring.api.model.SimpleBMOder
3:	23428922	6694953456	[Ljava.util.concurrent.ConcurrentHashMap\$HashEntry;
4:	11574497	5007170576	[C
5:	178973	2246627208	[B
6:	23357783	1681760376	java.util.concurrent.ConcurrentHashMap
7:	26115162	1253527776	java.util.HashMap\$Entry
8:	23386843	1122568464	java.util.concurrent.locks.ReentrantLock\$NonfairSync
9:	23381591	1122316368	java.util.concurrent.ConcurrentHashMap\$Segment
10:	27363594	1094543760	java.lang.String
11:	22350856	894034240	com.ullink.ulmonitoring.api.model.IndentedHelper
12:	22350849	894033960	com.ullink.ulmonitoring.api.model.OrderLine
13:	23357784	747639528	[Ljava.util.concurrent.ConcurrentHashMap\$Segment;
14:	22350792	715225344	com.ullink.ulmonitoring.extension.view.HierarchicalOrderLine
15:	23356168	560548032	com.ullink.ulmonitoring.api.model.property.PropertyHolderImpl
...			
817:	40	4480	com.ullink.ulmonitoring.extension.view.HierarchicalViewDataContainer

War stories

Client class (104 bytes)

com.ullink.oms.model.Client 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)	3b de 91 df (0011 1011 1101 1110 1001 0001 1101 1111)
12	4	String	OdisysObject.id	null
16	4	String	Client.type	null
20	4	Capacity	Client.capacity	null
24	4	String	Client.description	null
28	4	String	Client.currency	null
32	4	String	Client.parentClientId	null
36	4	Collection	Client.contact	null
40	4	String	Client.data	null
44	4	Contact	Client.office	null
48	4	String	Client.originCountry	null
52	4	Boolean	Client.allowUnknownAccount	null
56	4	String	Client.connectionDetails	null
60	4	String	Client.backoffice	null
64	4	Policy	Client.policy	null
68	4	ConfirmationSchedule	Client.confirmationSchedule	null
72	4	Collection	Client.stampExemptCountries	null
76	4	Map	Client.customProperties	null
80	4	Delivery	Client.delivery	null
84	4	Warehousing	Client.warehousing	null
88	4	Map	Client.externalIds	null
92	4	String	Client.feeId	null
96	4	Boolean	Client.active	null
100	4		(loss due to the next object alignment)	

Instance size: 104 bytes (estimated, add this JAR via -javaagent: to get accurate result)

Space losses: 0 bytes internal + 4 bytes external = 4 bytes total

War stories

num	#instances	#bytes	class name
1:	116925840	18595529736	[C
2:	100508918	10452927472	com.ullink.oms.model.Client
3:	116908590	2805806160	java.lang.String
4:	39690	773638968	[B
5:	77391	596989120	[Ljava.lang.Object;
6:	6116590	195730880	java.util.HashMap\$Entry
7:	2370792	149697448	[Ljava.util.HashMap\$Entry;
8:	2406042	115490016	java.util.HashMap
9:	149661	20530168	<constMethodKlass>
10:	149661	19166816	<methodKlass>

Event user.update

1 user with 32 560 Clients

3796 events = 245M Client instances => 25GB + Strings

Persistence delete operation keep objects in a Collection

War stories

Instrument class:

- 62 ref fields
- String, BigDecimal, Boolean, Double, Collection, Custom Classes (Alternateld (56B), Time (80B), Enums (24B)...)

Size: 264 bytes

War stories

Instrument Referential: 3M

Only Instrument instances: 792MB

Add 8 chars for instrumentId: +168MB ($56 \times 3M$)

Add 4 chars for symbol: + 312MB ($48 \times 3M + 56 \times 3M$)

Add settlement date: + 408MB ($56 \times 3M + 80 \times 3M$)

Add 3 chars for currency: + 138MB ($46 \times 3M$)

Total 1818MB

Other representation:

for each instrument : `Map<String, MDRecord>`

Solutions

Solutions

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

Measure, don't guess!

Kirk Pepperdine & Jack Shirazi

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>



Q & A

Jean-Philippe BEMPEL
Performance Architect

@jpbempel
<http://jpbempel.blogspot.com>