Создаём настоящее

Скажите “Ой!”

JVM и OOM Killer

- Я – Владимир Плизга́
- 2011-2021: **ЦФТ** (Java)
    - бэкенд Интернет-банков
- 2021-⏳: **Tibbo Systems** (Java/Kotlin)
    - бэкенд IoT-платформы
- 2023-⏳: **StegoText.ru** (Java/Kotlin)
    - прикладная криптография

Toparvion

toparvion.pro

StegoTrend

Случай из практики

- Обработка ≈20 метрик с 10K устройств
- Liberica JDK 8, G1 GC, -Xmx16G
- Oracle Linux, 24 GB RAM, no swap

Основные понятия

Чтобы говорить на одном языке

java.lang.OutOfMemoryError: Java heap space

Heap Space

Young Generation

Old Generation

Virtual

From
Survivor 0

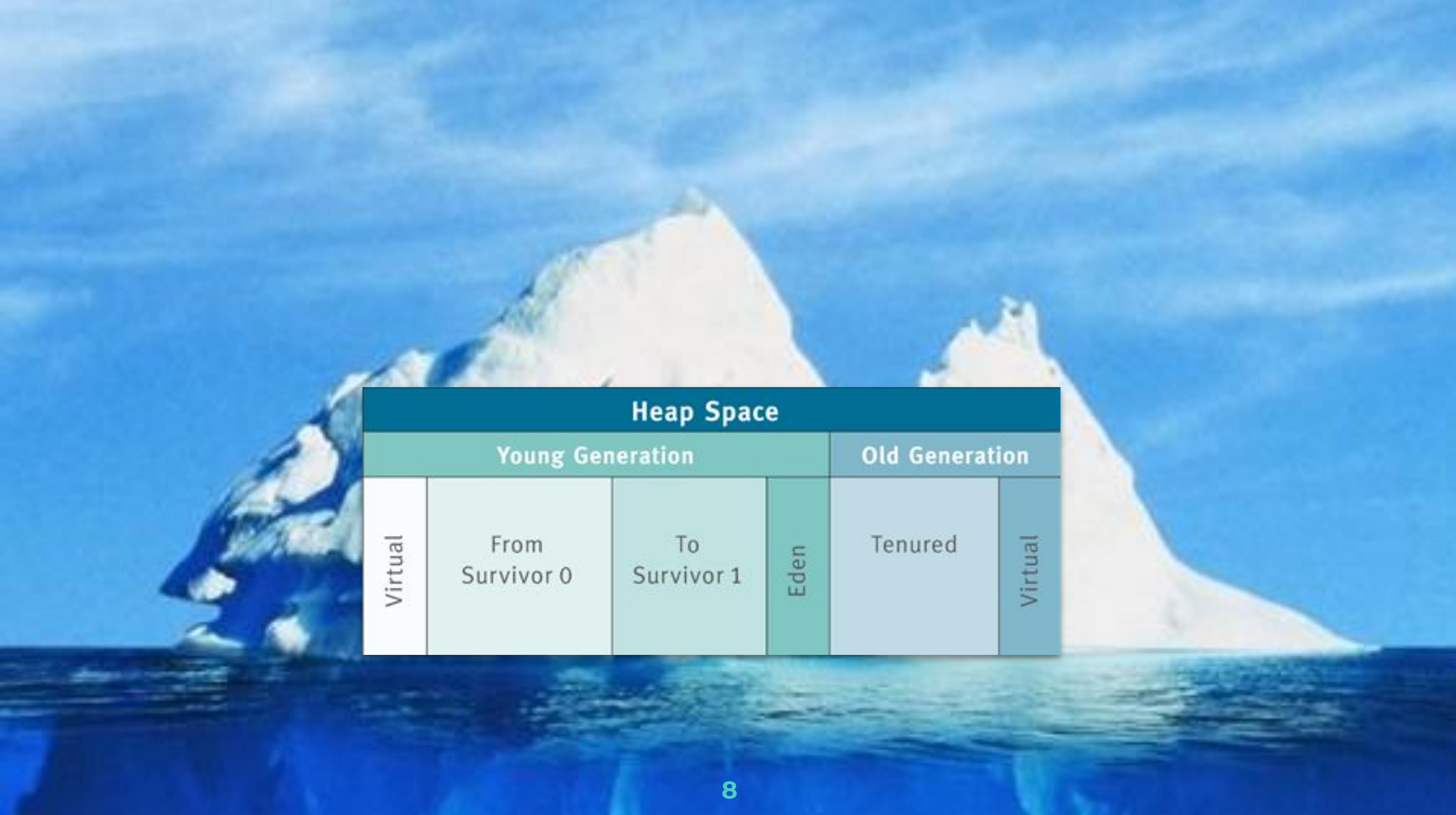
To
Survivor 1

Eden

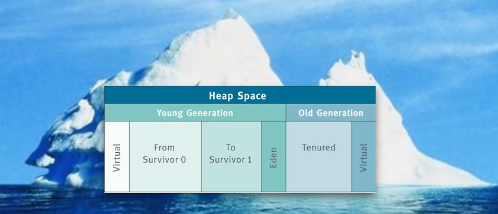
Tenured

Virtual

https://www.linkedin.com/pulse/java-virtual-machine-changes-78-9-kunal-saxena


Heap Space
Young Generation
Old Generation
Virtual
From
Survivor 0
To
Survivor 1
Eden
Tenured
Virtual

HEAP



Code Cache +
GC & Compiler +
Symbol tables +
Thread stacks +
Direct buffers +
Mapped files +
Metaspace +
Native libs +
malloc +
...

NON-HEAP
(off-heap, native)

https://www.youtube.com/watch?v=kKigibHrV5I

Что бывает при нехватке non-heap?

В лучшем (редком) случае:

```
java.lang.OutOfMemoryError: Direct buffer memory
```

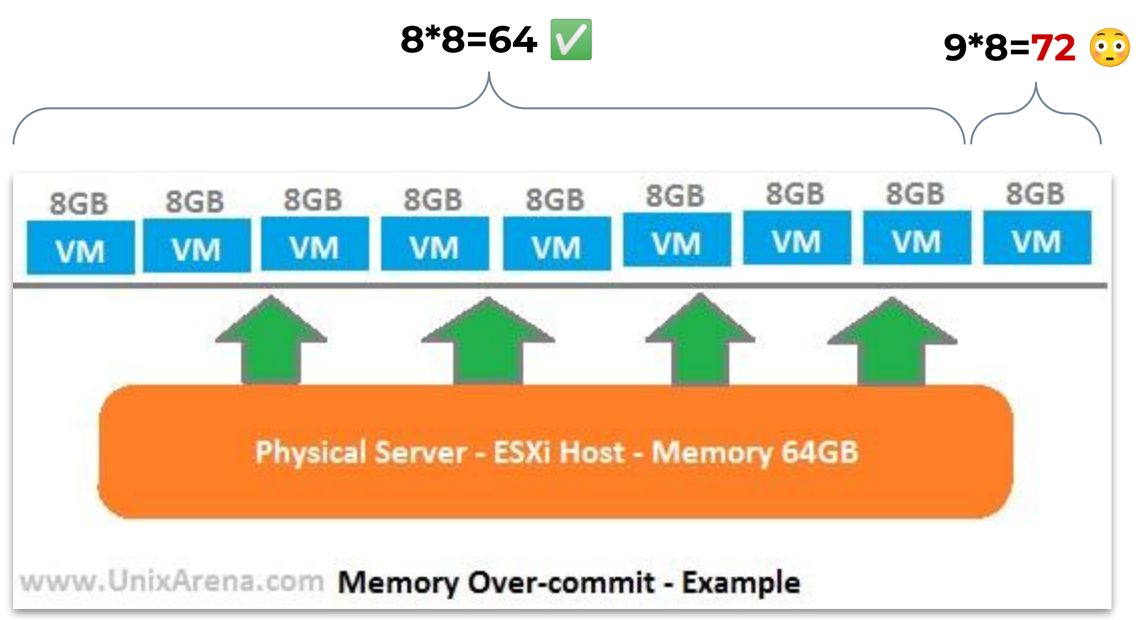
Но как правило: (логи ОС)

```
kernel: oom-kill:constraint=CONSTRAINT_NONE, ...
kernel: Out of memory: Killed process 3618718 (java)
        total-vm:28272408kB, anon-rss:22456024kB, ...
kernel: oom_reaper: reaped process 3618718 (java),
        now anon-rss:5949484kB, ...
```

А чё так грубо?

Нельзя ли просто выдать ошибку при излишнем запросе памяти?

Memory Overcommitment



https://www.unixarena.com/2014/07/how-vmware-vsphere-is-saving-physical-memory.html/

Memory Overcommitment

А если нет, то Out Of Memory Killer

- “Предотвращатель” большой катастрофы
- Не имеет явного выключателя
- Поведение зависит от многих факторов*

* Например, от **swap**, если он есть

Что можно успеть при OOM?

В куче

- ~~`try/catch`~~
- `-XX:+HeapDumpOnOutOfMemoryError`
- `-XX:OnOutOfMemoryError=cmd`

Вне кучи

© tlum.ru

Где искать первые признаки?

- В Linux: top, pmap, cat /proc/<pid>/status
- В JMX: jconsole, Mission Control, VisualVM
- В JVM: **Native Memory Tracking**

jconsole

Native Memory Tracking

Главная зацепка в расследовании

JVM NMT

- Встроенная функция JVM (в т.ч. HotSpot)
- Надо включить заранее:

```
-XX:NativeMemoryTracking=[off|summary|detail]
```

- Имеет overhead:
  - -5-10% performance
  - +2 words/malloc

https://docs.oracle.com/en/java/javase/21/vm/native-memory-tracking.html

Как получить результаты NMT

- Задать референтную точку: (опционально)

```
$ jcmd <pid> VM.native_memory baseline
```

- Запросить текущую статистику:

```
$ jcmd <pid> VM.native_memory summary[.diff]
```

- Если нет jcmd, можно через jattach:

```
$ jattach <pid> jcmd "VM.native_memory summary"
```

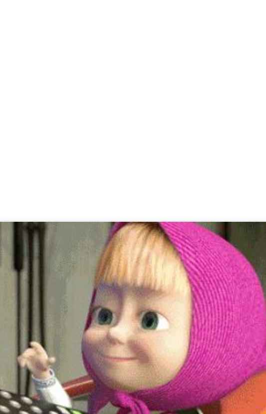
Diagnostic Commands

Operations

- GC.heap_info
- GC.run
- GC.run_finalization
- JFR.start
- Thread.print
- VM.cds
- VM.class_hierarchy
- VM.classloaders
- VM.metaspace
- VM.native_memory
- VM.print_touched_methods
- VM.stringtable
- VM.symboltable
- VM.systemdictionary
- GC.class_histogram
- JVMTI.data_dump

Name	Value	Description
baseline	<Boolean>	request runtime to baseline current memory usage, so it can be comp
detail	<Boolean>	request runtime to report memory allocation >= 1K by each callsite.
detail.diff	<Boolean>	request runtime to report memory detail comparison against previo
scale	<String>	Memory usage in which scale, KB, MB or GB
shutdown	<Boolean>	deprecated.
statistics	<Boolean>	print tracker statistics for tuning purpose.
summary	<Boolean>	request runtime to report current memory summary, which includes
summary.diff	<Boolean>	request runtime to report memory summary comparison against pre

Help | Execute | VM.native_memory



Mission Control — вариант jcmd

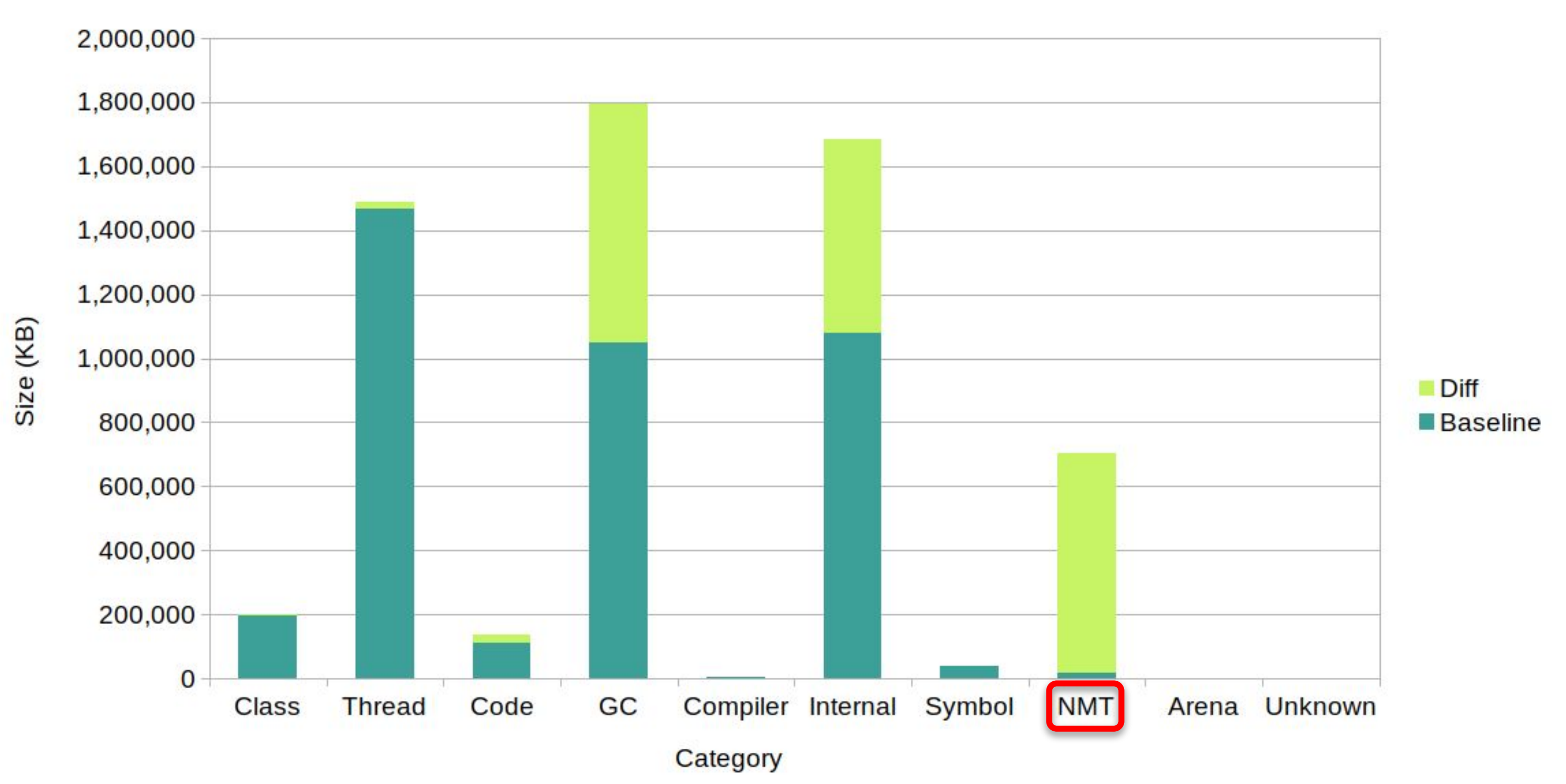
для ~~ленивых~~ экономных

Пример вывода

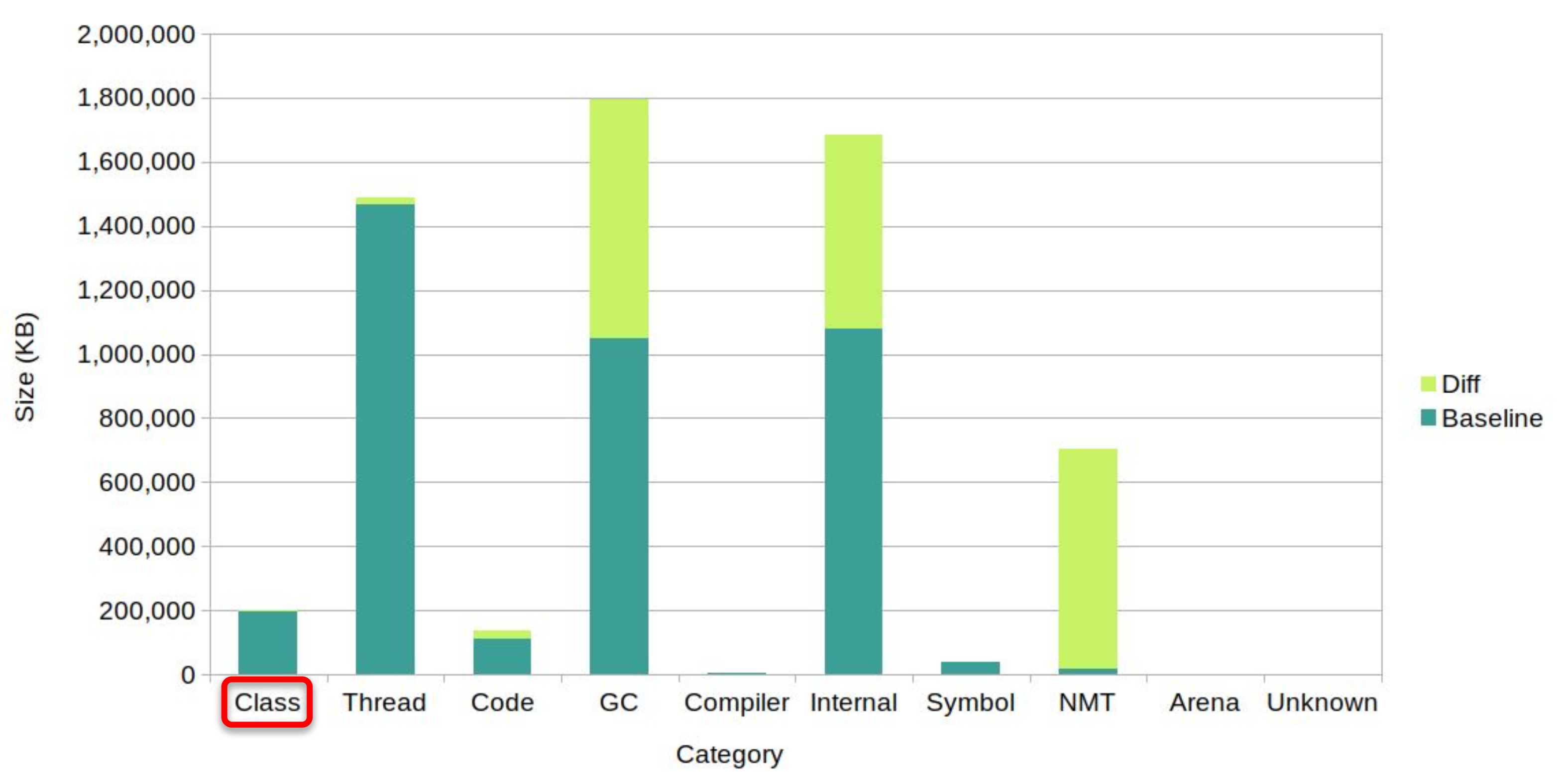
- Растёт почти с каждой версией JDK
- Категории описаны хоть и не полностью

```
Native Memory Tracking:

Total: reserved=22607111KB +1636088KB, committed=21415719KB +1656876KB

-                 Java Heap (reserved=15728640KB, committed=15728640KB)
                            (mmap: reserved=15728640KB, committed=15728640KB)

-                     Class (reserved=1223832KB +2491KB, committed=200116KB +3259KB)
                            (classes #30901 -280)
                            (malloc=5272KB +443KB #65472 +5996)
                            (mmap: reserved=1218560KB +2048KB, committed=194844KB +2816KB)

-                    Thread (reserved=1485308KB +18845KB, committed=1485308KB +18845KB)
                            (thread #3734 +48)
                            (stack: reserved=1468116KB +18624KB, committed=1468116KB +18624KB)
                            (malloc=12822KB +165KB #22398 +288)
                            (arena=4371KB +56 #7462 +96)

-                      Code (reserved=272270KB +3996KB, committed=135318KB +24016KB)
                            (malloc=22670KB +3996KB #28082 +3766)
                            (mmap: reserved=249600KB, committed=112648KB +20020KB)

-                        GC (reserved=1408163KB +315577KB, committed=1408163KB +315577KB)
                            (malloc=791715KB +315577KB #525565 +248761)
                            (mmap: reserved=616448KB, committed=616448KB)

-                  Compiler (reserved=5396KB +751KB, committed=5396KB +751KB)
                            (malloc=5262KB +751KB #7728 +1060)
                            (arena=135KB #7)

-                  Internal (reserved=1686946KB +603978KB, committed=1686942KB +603978KB)
                            (malloc=1686910KB +603978KB #45439096 +44579107)
                            (mmap: reserved=36KB, committed=32KB)

-                    Symbol (reserved=37270KB +35KB, committed=37270KB +35KB)
                            (malloc=32564KB +35KB #362454 +460)
                            (arena=4707KB #1)

-    Native Memory Tracking (reserved=727516KB +701253KB, committed=727516KB +701253KB)
                            (malloc=1341KB +539KB #16736 +6500)
                            (tracking overhead=726175KB +700714KB)

-               Arena Chunk (reserved=1050KB -10839KB, committed=1050KB -10839KB)
                            (malloc=1050KB -10839KB)

-                   Unknown (reserved=30720KB, committed=0KB)
                            (mmap: reserved=30720KB, committed=0KB)
```

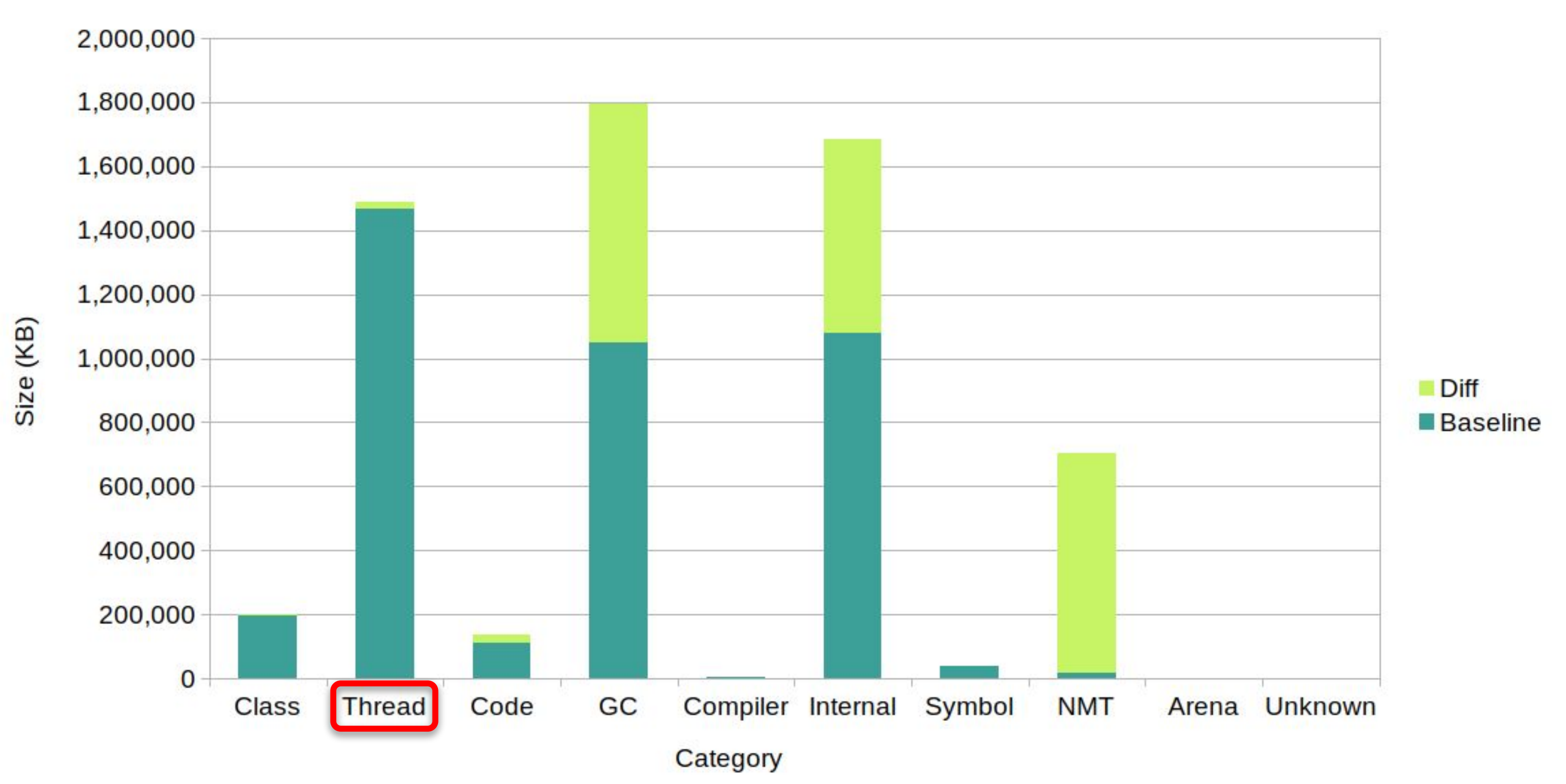
Пример категорий NMT



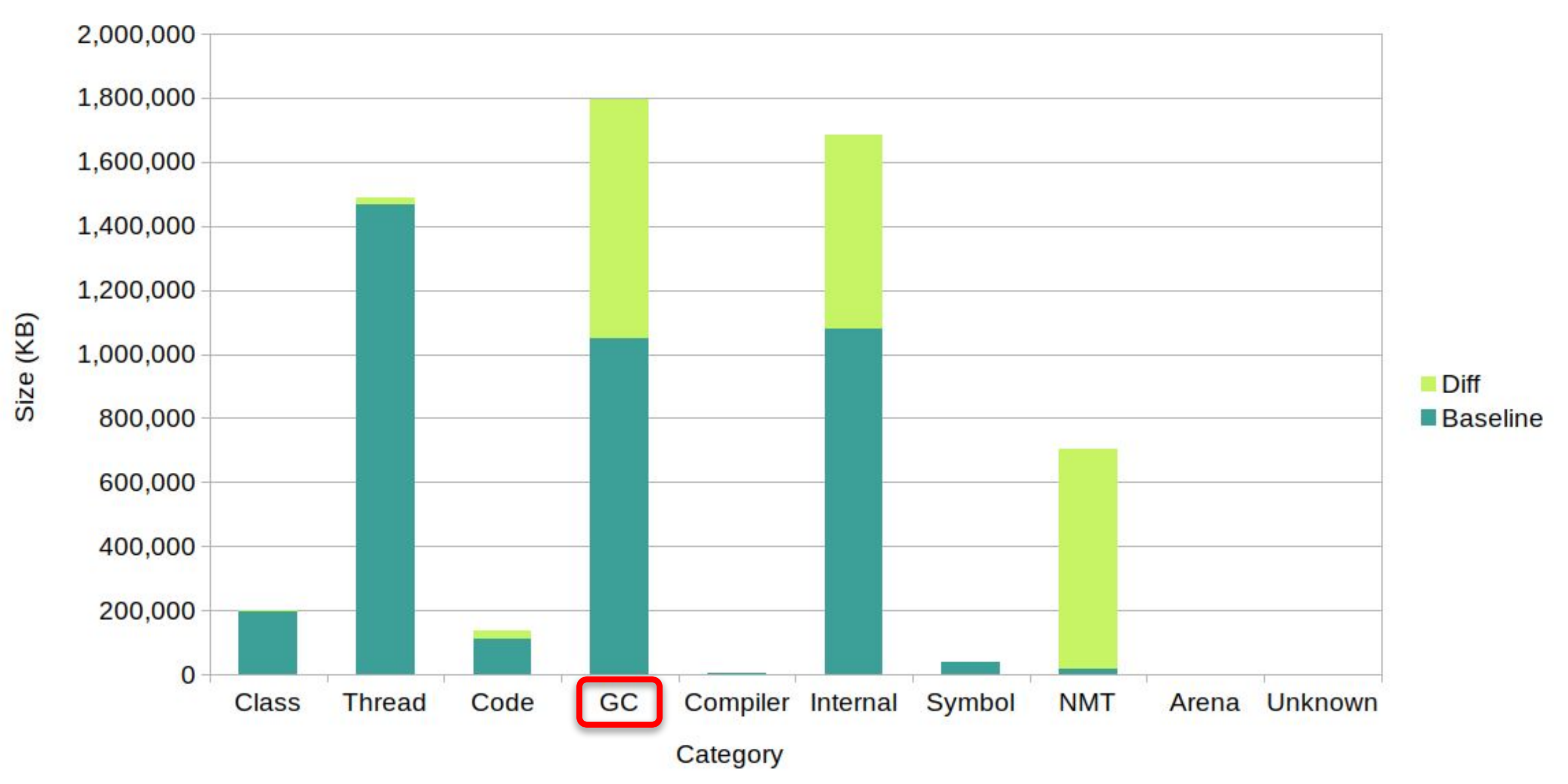
Пример категорий NMT



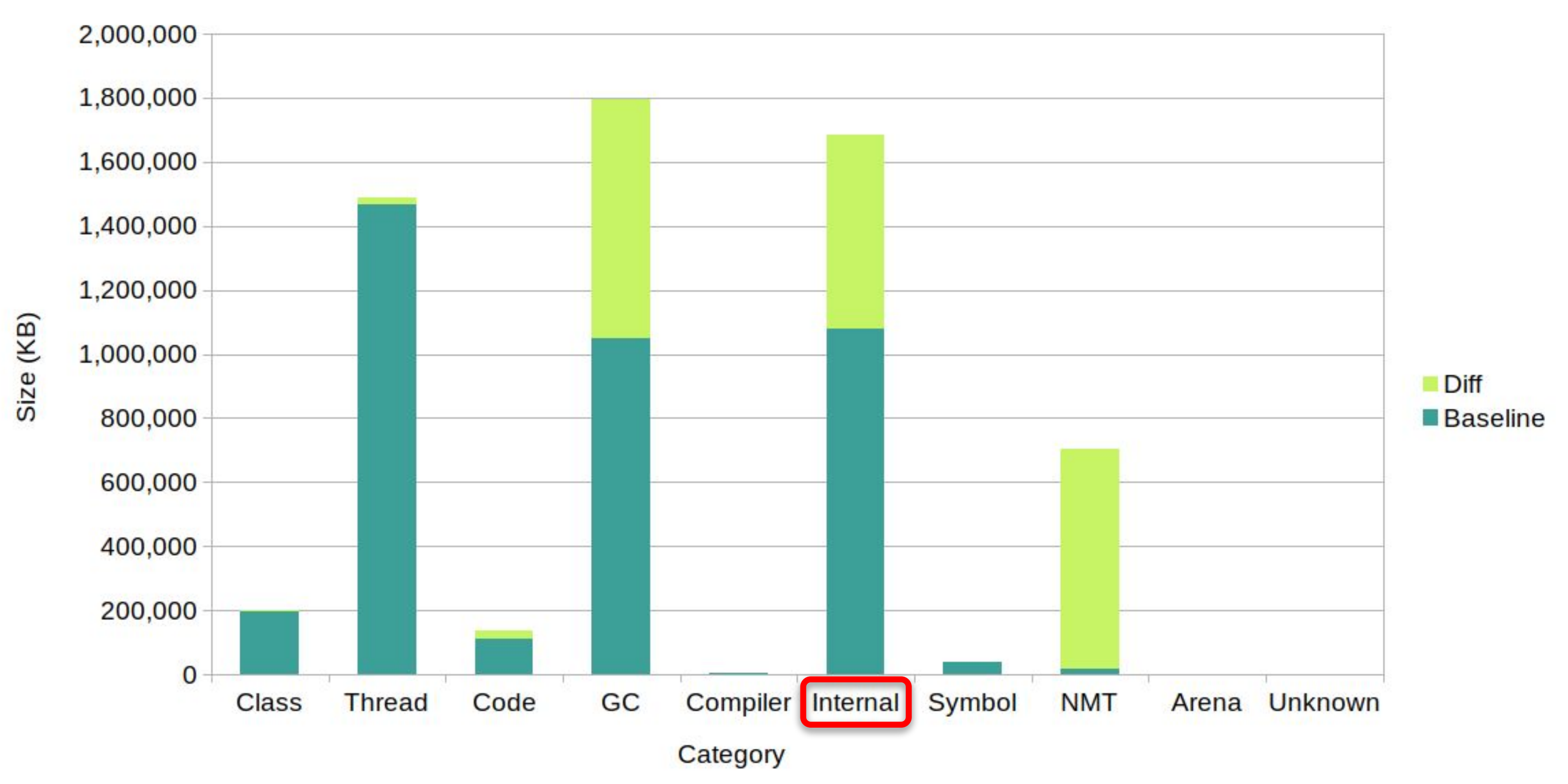
Пример категорий NMT



Пример категорий NMT



Пример категорий NMT



NMT: категория OTHER (INTERNAL)

- По смыслу означает “прочее”
  - поэтому в Java 11 разбита на две (+Other)
- Не имеет ограничений
  - кроме частичного `-XX:MaxDirectMemorySize`
- Может содержать почти что угодно
  - поэтому нужны детали NMT

Пример деталей для INTERNAL

```
[0x00007fc009a30ab5] jni_GetByteArrayElements+0x165
        (malloc=2263KB type=Internal +53KB
                                         #579408 +13550)

# (спустя несколько дней)
[0x00007fc009a30ab5] jni_GetByteArrayElements+0x165
        (malloc=282704KB type=Internal +280493KB
                                         #72372172 +71806314)
```

Где ещё почитать о категориях

- JVM Anatomy Quark #12: Native Memory Tracking
- Oracle Troubleshooting Guide – NMT
- Java using much more memory - StackOverflow

NMT описывает всю нативную память

(нет)

Прочие источники аллокаций

Сравнение показаний JVM и ОС

```
$ jcmd `pgrep java` VM.native_memory summary | grep Total
Total: reserved=23121490KB, committed=21934730KB

$ sudo cat /proc/`pgrep java`/status | grep RSS
VmRSS: 22520924 kB
```

- Разница: 22520924-21934730=586184 KB
  - Где ещё **500+МБ** ?

Java ByteBuffers

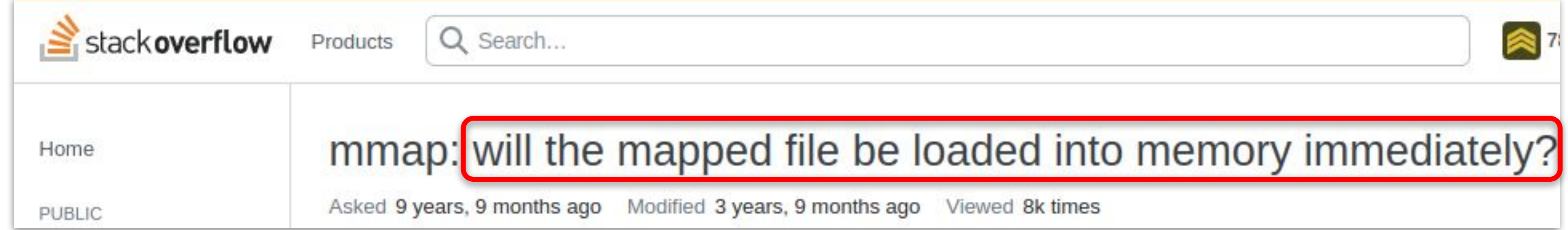
- Последовательности целых чисел для быстрого ввода-вывода
- Бывают:
  - non-direct (в куче)
  - direct (вне кучи)
    - mapped

      Для отображения файлов в память.

Как NMT учитывает ByteBuffers

- `ByteBuffer.allocateDirect()` ✅
  - учитывается в **Other** (до JDK 11 — **Internal**)
- `ByteBuffer.allocate()` ✅
  - учитывается в **Heap**
- `FileChannel.map()` ❌
  - опаньки

https://stackoverflow.com/questions/62635023/native-memory-consumed-by-jvm-vs-java-process-total-memory-usage



4 Answers

42

No, yes, maybe. It depends.

https://stackoverflow.com/questions/19653618/mmap-will-the-mapped-file-be-loaded-into-memory-immediately

Сколько занимает mapped

```
$ pmap `pgrep java`
      Address    Size  Mapping
55d98e400000        4  java
...
7fbf25e00000 204800  CommitLog-6-1691993268689.log
7fbf32600000   2048
7fbf32800000 204800  CommitLog-6-1691993268688.log
```

Сколько реально занимает mapped

```
$ pmap -X `pgrep java`
     Address   Size      Rss Mapping
55d98e400000      4        4 java
...
7fbf25e00000 204800        4 CommitLog-6-1691993268689.log
7fbf32600000   2048     2048
7fbf32800000 204800     1644 CommitLog-6-1691993268688.log
```

Что нужно знать об mapped buffers

- Mapped-файл не обязательно занимает в памяти столько, сколько весит сам
- “Отработанные” части файла могут составлять заметную часть buffer cache
  - но это, как правило, не проблема

Что нужно знать об mapped buffers

- Опасаться лучше не среднего, а всплесков

Пример “частокола”, провоцировавшего OOM Killer

Плагин **Buffer Pools** для VisualVM

Mapped

Memory Used: 419 430 400 B **Total Capacity:** 419 430 400 B
Count: 2

2 600 MB
2 500 MB
2 400 MB
2 300 MB
2 200 MB
2 100 MB
2 000 MB
1 900 MB
1 800 MB
1 700 MB
1 600 MB
1 500 MB
1 400 MB
1 300 MB
1 200 MB
1 100 MB
1 000 MB
900 MB
800 MB
700 MB
600 MB
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB

16:55 17:00 17:05 17:10 17:15 17:20 17:25 17:30

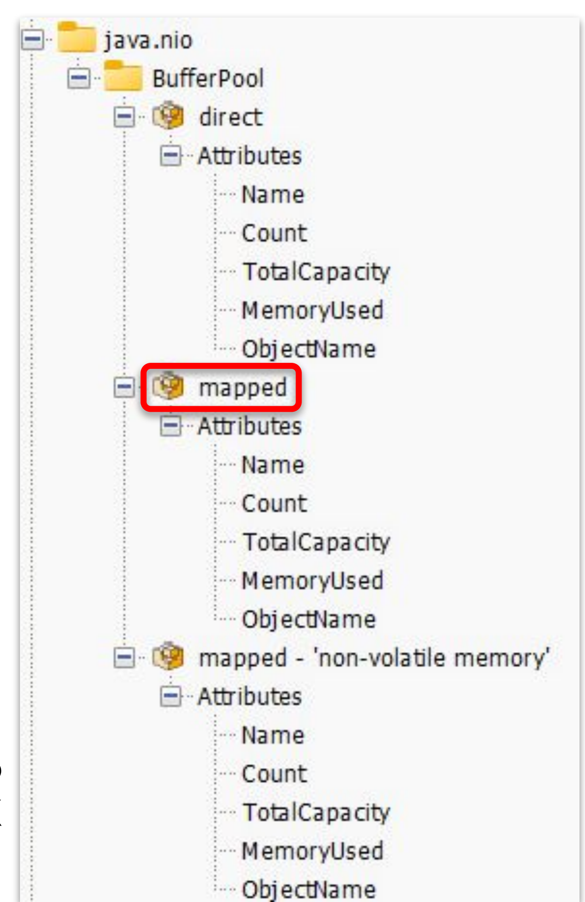
Activate Windows
Go to Settings to activate Windows.

Где ещё посмотреть

Для работы с JMX из консоли есть утилитка Jmxterm:

docs.cyclopsgroup.org/jmxterm

java.nio
JMX
MBean



Найти источник (через async-profiler)

1. Начать запись **mmap**-аллокаций:

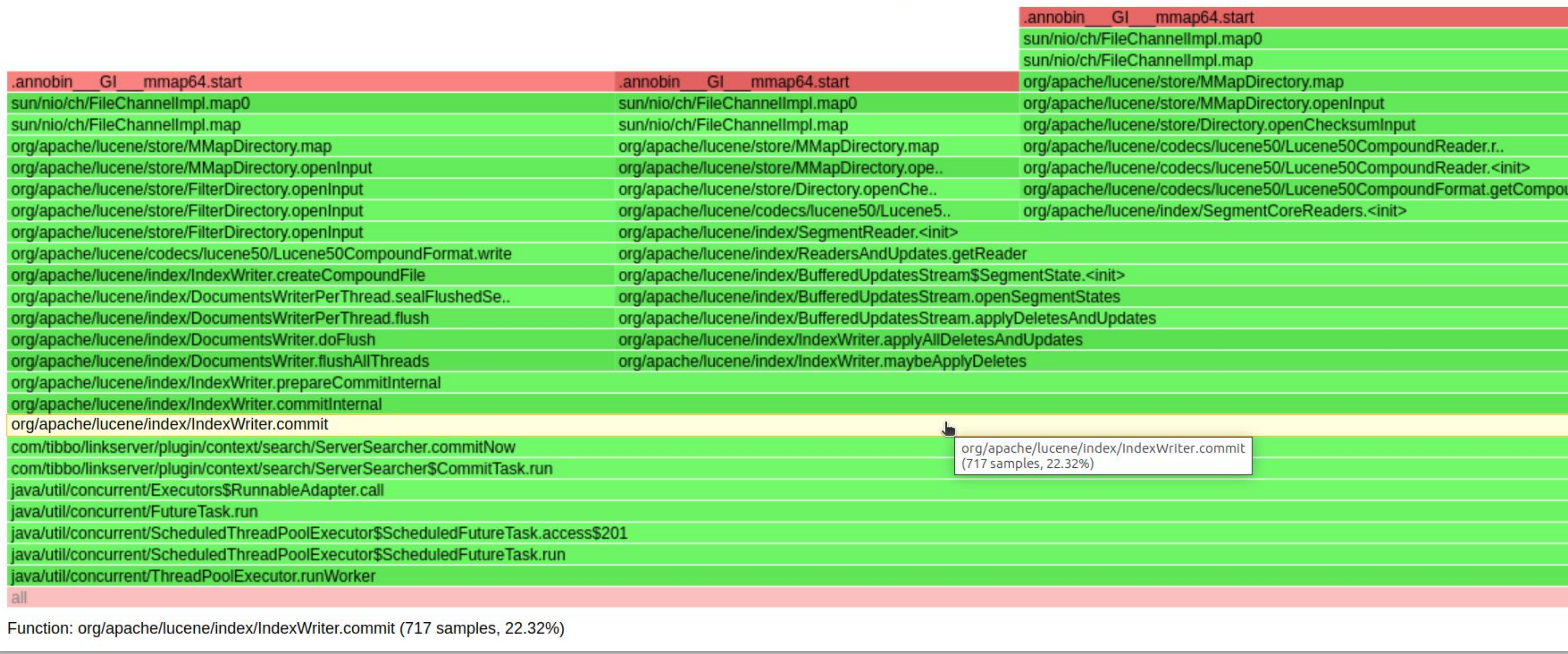
```
$ sudo ./profiler.sh start -e mmap `pgrep java`
```

2. Узнать, как дела (если надо):

```
$ sudo ./profiler.sh status `pgrep java`
```

3. Сохранить текущий результат:

```
$ sudo ./profiler.sh dump -f mmap.html `pgrep java`
```



Пример обнаружения пиков mmap-аллокаций в Apache Lucene

NMT doesn't give complete information for the memory used by the **class data sharing** (CDS) archive.

https://docs.oracle.com/en/java/javase/17/vm/native-memory-tracking.html

Что нужно знать о CDS

- Немного ускоряет запуск JVM
- Экономит память, если в системе несколько процессов одной и той же JVM
- Со временем по памяти не растёт
- Можно отключить: `-Xshare:off`
- Подробнее — в этой статье

Если вся нативная память ОК, то OOM Killer не придёт

(нет)

Механизм аварийного вытеснения

The OOM killer selects
a task to sacrifice
for the sake of
the overall system health.

https://docs.kernel.org/admin-guide/mm/concepts.html?highlight=killer#id9

Java не всегда виновата

```
$ grep “invoked oom-killer” messages-20230702
kernel: dnf invoked oom-killer
kernel: new_free_mem_me invoked oom-killer
kernel: VM Thread invoked oom-killer
kernel: mem_nonheap_tes invoked oom-killer
kernel: InConsumerThrea invoked oom-killer
kernel: snmpd invoked oom-killer
kernel: MutationStage-8 invoked oom-killer
```

Почему так?

```
1097    /**
1098     * out_of_memory - kill the "best" process when we run out of memory
1099     * @oc: pointer to struct oom_control
1100     *
1101     * If we run out of memory, we have the choice between either
1102     * killing a random task (bad), letting the system crash (worse)
1103     * OR try to be smart about which process to kill. Note that we
1104     * don't have to be perfect here, we just have to be good
1105     */
1106    bool out_of_memory(struct oom_control *oc)
1107    {
```

https://github.com/torvalds/linux/blob/c1a515d3c0270628df8ae5f5118ba859b85464a2/mm/oom_kill.c#L1106

Алгоритм выбора жертвы

```
226         /*
227          * The baseline for the badness score is the proportion of RAM that each
228          * task's rss, pagetable and swap space use.
229          */
230         points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
231                 mm_pgtables_bytes(p->mm) / PAGE_SIZE;
232         task_unlock(p);
233 
234         /* Normalize to oom_score_adj units */
235         adj *= totalpages / 1000;
236         points += adj;
237 
238         return points;
```

https://github.com/torvalds/linux/blob/c1a515d3c0270628df8ae5f5118ba859b85464a2/mm/oom_kill.c#L230

Алгоритм выбора жертвы

1. Рассчитать “очки” для каждого процесса:

   `points = RSS + page_table + swap`

2. Нормализовать к `1000`
3. Принять поправку на `oom_score_adj`
4. Взять процесс с максимальным `score`
5. `SIGKILL`

https://github.com/torvalds/linux/blob/c1a515d3c0270628df8ae5f5118ba859b85464a2/mm/oom_kill.c#L230

И причём здесь Java?

```
$ ./oom-score-list.sh
PID     Score Process
988     0     irqbalance
99      0     writeback
...
992     0     chronyd
657     1     systemd-journal
269670  898   java
```

https://www.baeldung.com/linux/memory-overcommitment-oom-killer

Что можно сделать

Что можно сделать

- Узнать текущий oom_score:

```
$ cat /proc/`pgrep java`/oom_score
```

- Поправить поправку (от -1000 до 1000):

```
$ sudo echo -100 > /proc/`pgrep java`/oom_score_adj
```

- Переключить поведение Killer'a:

```
$ sudo sysctl -w vm.oom_kill_allocating_task=1
```

Если Killer’a всё-таки вызвала java

```
$ grep “invoked oom-killer” messages-20230702
kernel: dnf invoked oom-killer
kernel: new_free_mem_me invoked oom-killer
kernel: VM Thread invoked oom-killer
kernel: mem_nonheap_tes invoked oom-killer
kernel: InConsumerThrea invoked oom-killer
kernel: snmpd invoked oom-killer
kernel: MutationStage-1 invoked oom-killer
```

Если Killer’a всё-таки вызвала java

- В логе ядра будет виден поток, а не процесс
- Название потока обрезается на 15 символах
- PID потока не равен PID процесса java
- Стектрейс не отражает состояние потока
- Подробнее — здесь (serverfault.com)

Выводы
Резюме
Ссылки

Ради чего
всё это
продолжалось

Takeaways

Предотвращение

- Делать запас памяти вне кучи (от 10%)
- Выяснять, какие сторонние либы работают в non-heap

Анализ

- NMT (jcmd)
- async-profiler
- pmap/top
- JMX/JFR

Устранение

- Фиксить утечки
- Сокращать ненужные кэши
- Менять аллокатор

Резюме

- Общего лимита нативной памяти нет
- Если пришёл OOM Killer — включай NMT
- Работу Killer'а надо не настраивать, а предотвращать

Спасибо!

Вопросы?

Владимир Плизга́

Бонусный раздел

Для тех, кому
всё-таки
не хватило

А если поменять аллокатор памяти?

- Функция malloc() в *nix ОС может иметь разные имплементации
- Их можно “подкладывать” на этапах:
  - компиляции
  - сборки/линковки
  - в runtime (подходит для JVM)

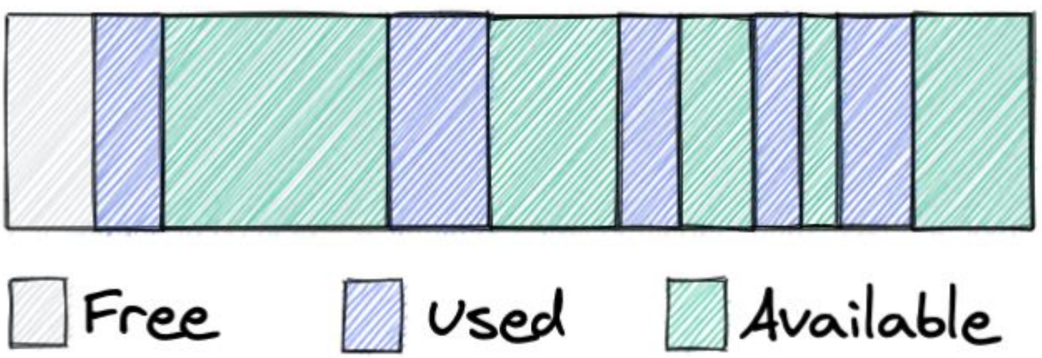
Что не так с обычным malloc? (glibc)

```
$ pmap `pgrep java` | grep -B 1 " 65.*K .* [ anon ]"
00007f637c000000    132K rw---   [ anon ]
00007f637c021000  65404K -----   [ anon ]
00007f6380000000    132K rw---   [ anon ]
00007f6380021000  65404K -----   [ anon ]
...
$ cat /proc/`pgrep java`/status | grep RssAnon
RssAnon:   1416720 kB
```

≈100 штук

Что не так с обычным malloc? (glibc)

- Может сильно фрагментировать память,
  - особенно в многопоточных приложениях



https://blog.cloudflare.com/the-effect-of-switching-to-tcmalloc-on-rocksdb-memory-use/

Альтернатива — Jemalloc

- Открытый аллокатор родом из FreeBSD (2005)
- Заточен под многопоточные приложения
- Имеет встроенные средства отладки
- Применяется в Facebook, Firefox, Mac OS X, …

Как подключить Jemalloc к JVM

1. Скачать и собрать
2. Выставить переменную окружения:
   `export LD_PRELOAD=/path/to/libjemalloc.so`
3. Выставить переменную окружения:
   `export MALLOC_CONF=prof:true,…` (опционально)
4. (пере)Запустить JVM

Jemalloc в действии

```
$ pmap `pgrep java` | grep -B 1 " 500K .* [ anon ]"
00007fbefd280000     12K -----   [ anon ]
00007fbefd283000    500K rw---   [ anon ]
00007fbefd300000     12K -----   [ anon ]
00007fbefd303000    500K rw---   [ anon ]
...
$ cat /proc/`pgrep java`/status | grep RssAnon
RssAnon:   1296356 kB
```

(Application) Class Data Sharing

- Фича JVM для ускорения запуска и уменьшения потребления памяти
- Как проверить, включена ли:

```
$ jcmd `pgrep java` VM.info | grep CDS
CDS archive(s) not mapped
CDS: off
```

- До Java 12 по умолчанию выключена

CDS для тех, кому за 12

```
$ jcmd `pgrep java` VM.info | grep CDS
CDS archive(s) mapped at:
[0x0000000800000000-0x0000000800be1000-0x0000000800be1000), size 12455936, SharedBaseAddress:
0x0000000800000000, ArchiveRelocationMode: 0.
CDS: on
```

Выясняем источник (JDK 17)

```
$ pmap `pgrep java` | grep jsa
00000007ffb00000    472K rw--- classes.jsa
00000007ffc00000    520K rw--- classes.jsa
0000000800000000   4448K rw--- classes.jsa
0000000800458000   7716K r---- classes.jsa

$ du -h $JAVA_HOME/lib/server/classes.jsa
14M  lib/server/classes.jsa
```

Список классов вшит в JDK

```
$ cat $JAVA_HOME/lib/classlist | head -10
# NOTE: Do not modify this file.
…
java/lang/Object
java/io/Serializable
java/lang/Comparable
java/lang/CharSequence
java/lang/constant/Constable
```

*Всего: **1401** класс (для JDK 17.0.7)*

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Sticker packs by:
  - https://t.me/addstickers/masha_mishka
  - https://t.me/addstickers/MASHAANDTHEBEARDIGE44