ТЕХНОАТОМ

Java

Java GC

Александр Помосов

Reminder

ТЕХНОАТОМ

Отметьтесь на портале

Reminder

ТЕХНОАТОМ

Обновите репозиторий

Agenda

TEXHOATOM

- Heap, Object layout
- Garbage Collectors
- Performance
- Interview questions

Agenda

TEXHOATOM

- Heap, Object layout
- Garbage Collectors
- Performance
- Interview questions

HotSpot JVM

TEXHOATOM

The following material is mostly JVM-specific

We will look at **HotSpot** implementation, as it is *De facto* standard for **servers**
http://www.oracle.com/technetwork/articles/javase/index-jsp-136373.html

Set apart Dalvik, ART (Android JVM)

Structure of java process

TEXHOATOM

Structure of java process is defined by Java Virtual Machine Specification (JVMS)
https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.3

Heap space	Method area
All the objects, created in program are stored here Heap structure depends on chosen Garbage Collector **(GC)**	Metaspace
	Runtime Constant Pool
	Field and Method Data
	Code for Methods and Constructors

Method area is mostly constant-size
The most dynamic (and bigger) part is **Heap**

Object allocation

TEXHOATOM

… So objects are allocated on heap

Q: Can objects (e.g. local for method) be allocated on stack?
A: Not specified. But currently not. For curious:
http://dev.cheremin.info/2016/02/stack-allocation-vs-scalar-replacement.html

Q: Is that this heap? https://en.wikipedia.org/wiki/Heap_(data_structure)
A: No! Heap has complex structure which is depends on chosen GC

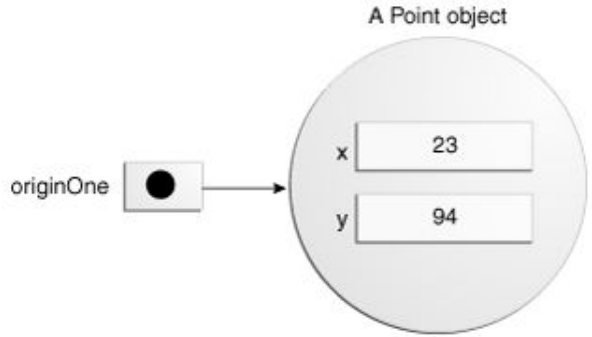
Q: So, how is Object represented in heap?
A: …

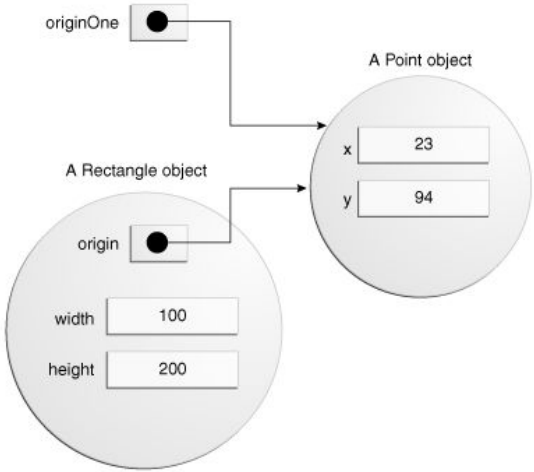
Object allocation - example

TEXHOATOM

https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html

`Point originOne = new Point(23,94);`	`Rectangle rectOne = new Rectangle(originOne, 100, 200);`
Meanwhile somewhere in heap...	Meanwhile somewhere in heap...





Game: Guess size

TEXHOATOM

By size we mean **footprint**, i.e. total size of reference + shadow size (all the referenced objects and primitives recursively)

What is the footprint of:

1. int x = 10;
2. new Integer(10);
3. new Long(10);
4. new Integer[1000];
5. new ArrayList<Integer>(1000);
6. new LinkedList<Integer>(1000);
7. new HashSet<Integer>(1000);

Warning: Not an easy game!

Object layout

TEXHOATOM

Layout - how objects are represented in heap

“The Java Virtual Machine does not mandate any particular internal structure for objects”
https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.7

We will look at HotSpot implementation (Java 8)

We need to go deeper...

Object layout details

TEXHOATOM

Object layout is:

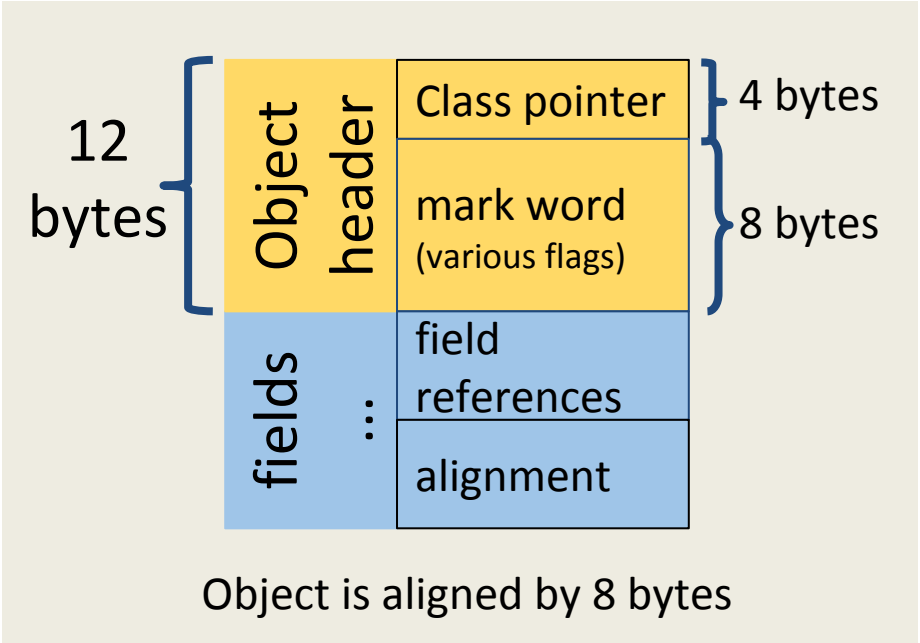
- JVM specific
- bitness dependent
- tunable by JVM flags

Example for **x64** system with

-XX:+UseCompressedOops (default)

for heaps less than 32GiB object reference is 32 bit

else 64 bit



http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html#compressedOop

jol

TEXHOATOM

Good tool for object layout analysis - **Java Object Layout**
http://openjdk.java.net/projects/code-tools/jol/

```
<dependency>
    <groupId>org.openjdk.jol</groupId>
    <artifactId>jol-core</artifactId>
    <version>put-the-version-here</version>
</dependency>
```

```
@see jol.samples.JOLSample_01_Basic
@see jol.samples.JOLSample_02_Alignment
@see jol.samples.JOLSample_03_Packing
@see jol.GuessSize
```

Agenda

TEXHOATOM

- Heap, Object layout
- Garbage Collectors
- Performance
- Interview questions

Garbage Collector

TEXHOATOM

Q: Why we need Garbage Collector
A: To remove garbage, obviously

Q: What is garbage?
A: Objects that are no longer referenced

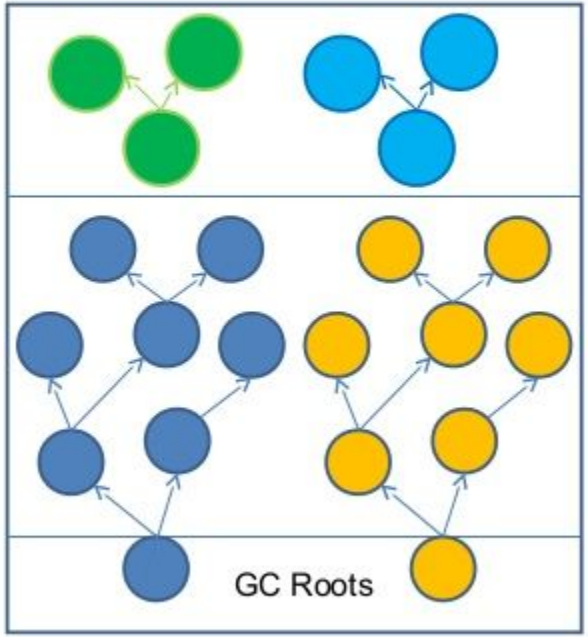
Q: Does cyclic references stall objects in heap forever
A: No. Object is garbage if it can not be reached from GC Roots

Q: What are GC Roots?
A: … (proceed reading)

GC Roots and Garbage

TEXHOATOM

- GC Roots
  - Class and static variables - class loaded by system class loader.
  - Thread - live thread
  - Stack Local - local variable or parameter of Java method
  - JNI Local - local variable or parameter of JNI method
  - JNI Global - global JNI reference
  - Monitor Used - objects used as a monitor for synchronization
  - Held by JVM - objects held from garbage collection by JVM for its purposes.



Problem 1. Memory Leaks

TEXHOATOM

GC behaves as expected unless you deal with low-level details via Unsafe or JNI.
Object, that is used somehow will never be deleted

The opposite problem is **memory leak** - when some object is no longer used, but still not counted as garbage. This is much common, but not as common as in c++.

Memory Leak Sources

TEXHOATOM

Thank to GC it is much harder to introduce memory leak in java than, e.g. in C++

- ObjectInputStream and ObjectOutputStream
  call reset() to flush inner object caches
- Threads (every thread has stack)
  use ThreadPools to control and reuse Threads
- non-static inner classes
  instance of non-static inner class has reference to instance of enclosing class
- thread-locals
  Thread-locals live while thread live unless explicitly cleared

Memory leaks examples:

http://stackoverflow.com/questions/6470651/creating-a-memory-leak-with-java
https://habrahabr.ru/post/132500/
(except String.substring() - this is not the case anymore)

Problem 2. GC Performance

TEXHOATOM

All GCs in HotSpot are **‘Stop-the-world’**
i.e. there are moments when all the application threads are stopped and GC is working.

Different GCs implement different strategies to reduce pauses. Some even give guarantees of maximum pause time.

There is an attempt to implement ‘ultra-low pause’ GC
http://openjdk.java.net/projects/shenandoah/ (not production-ready)
There are JVM implementations where GC is pauseless:
https://www.azul.com/products/zing/ (proprietary)

GC Tuning Capabilities

TEXHOATOM

GC is not controlled directly (there is no legal way to force GC)
but is managed by JVM

```
System.gc(); // this is just recomendation
Runtime.gc();// this is just recomendation
```

Java provides mechanisms to tune GC via JVM parameters

You can choose:

- Garbage collector implementation
- Several Garbage collector parameters

Idea behind GC Implementations

ТЕХНОАТОМ

Most objects
live fast, die young

New object
most likely die
immediately

If object survives,
it will probably
live long

‘Generations’ Idea

TEXHOATOM

We can divide objects into

- **young generation**
  (newly created objects)
  here we achieve high throughput - every GC removes almost all objects
- **survivor**
  objects that survived several GCs
- **old generation**
  (objects that survived many garbage collections)
  here we achieve memory efficiency - GCs are rare, memory is not fragmented

This idea is used in every GC implementation in HotSpot. It works!

Available Garbage Collectors

TEXHOATOM

- **Serial** (последовательный)

  simple, suitable for single-threaded applications with small heap
- **Parallel** (параллельный)

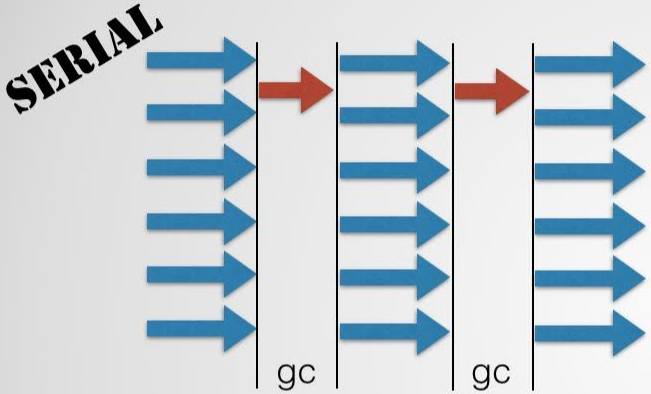
  same as serial, but adds parallelism to some GC stages and is able to adapt to given performance goals
- **Concurrent Mark Sweep** (CMS)

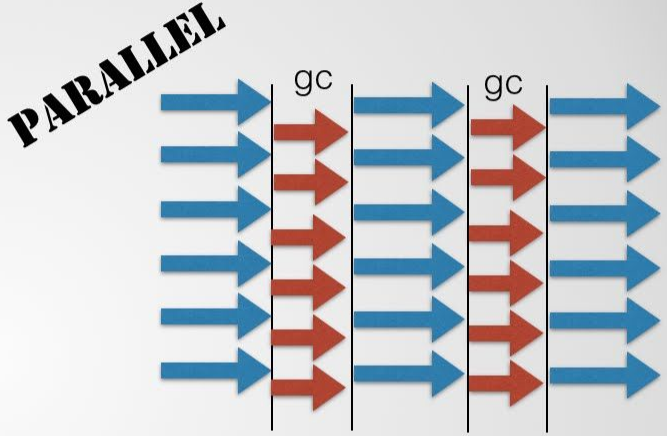
  minimize maximum pause time, is suitable for large heaps
- **Garbage-First (G1)**

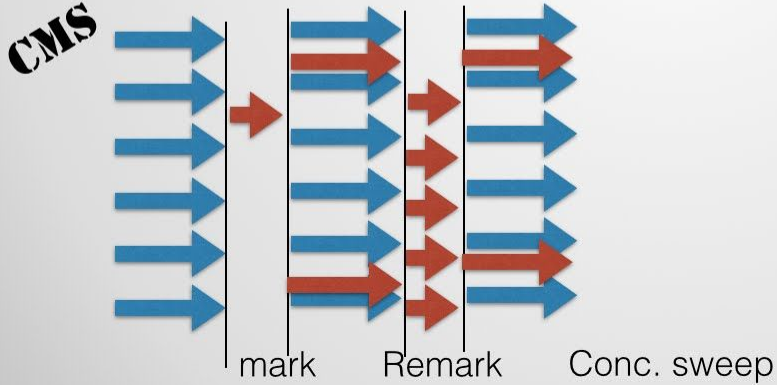
  In perspective will replace CMS. Has different heap structure. Suitable for highly concurrent applications with big heaps

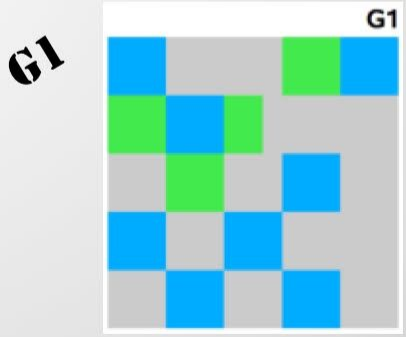
https://habrahabr.ru/post/269621/
https://habrahabr.ru/post/269707/
https://habrahabr.ru/post/269863/

Garbage Collectors





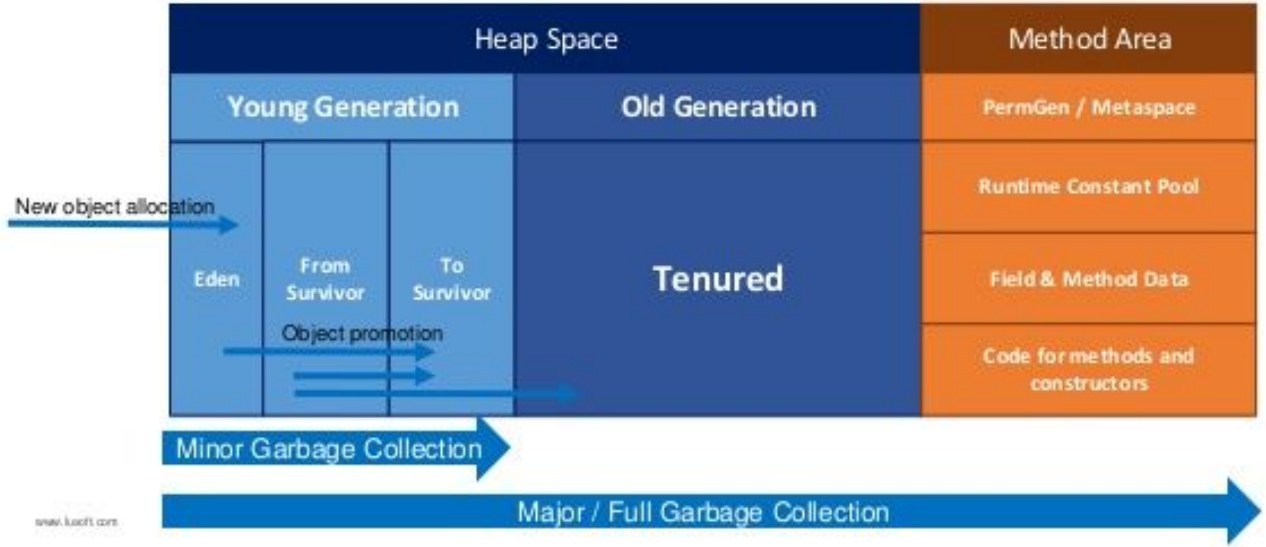




Serial/Parrallel/CMS Heap Structure

TEXHOATOM

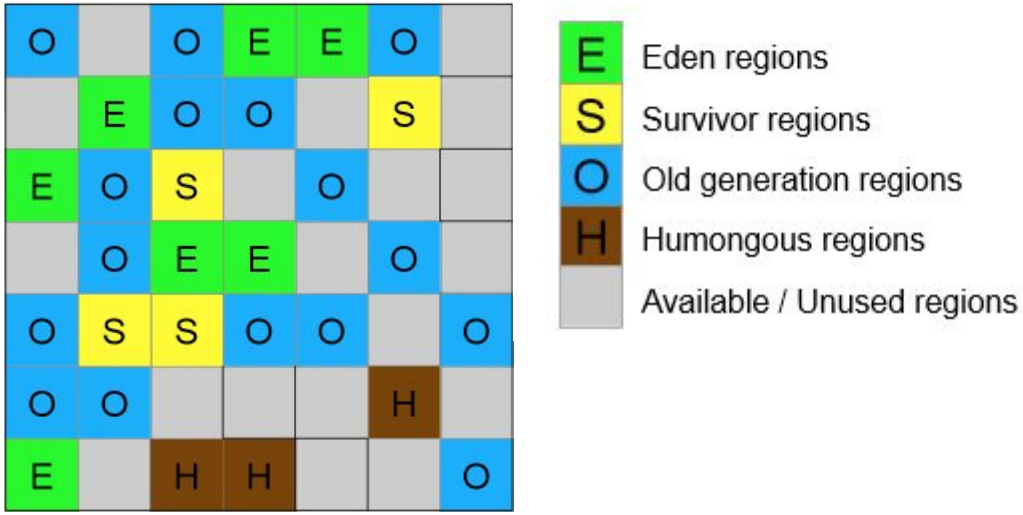
Exact heap structure is defined by chosen GC. For all standard HotSpot GC except G1GC (Serial, Parallel, Concurrent Mark Sweep) heap structure is as following:



G1GC

TEXHOATOM

divides heap into regions
that can act as
Eden, Survival or Old
generation regions
and it can change role
dynamically



http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html
http://www.oracle.com/technetwork/articles/java/g1gc-1984535.html
https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/g1_gc.html

Trace objects with JOL

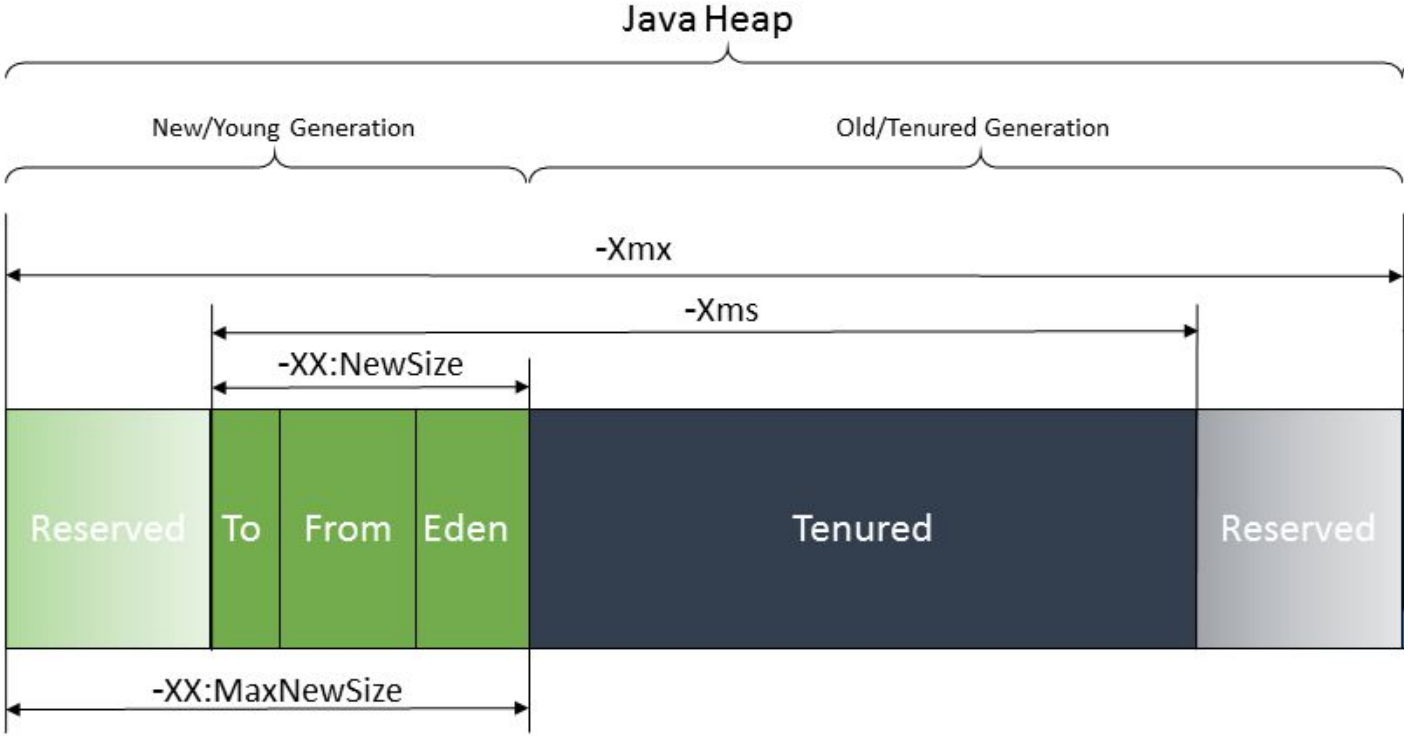
TEXHOATOM

```
New object allocation
@see jol.samples.JOLSample_17_Allocation
Promotion
@see jol.samples.JOLSample_19_Promotion
```

Sizing generations

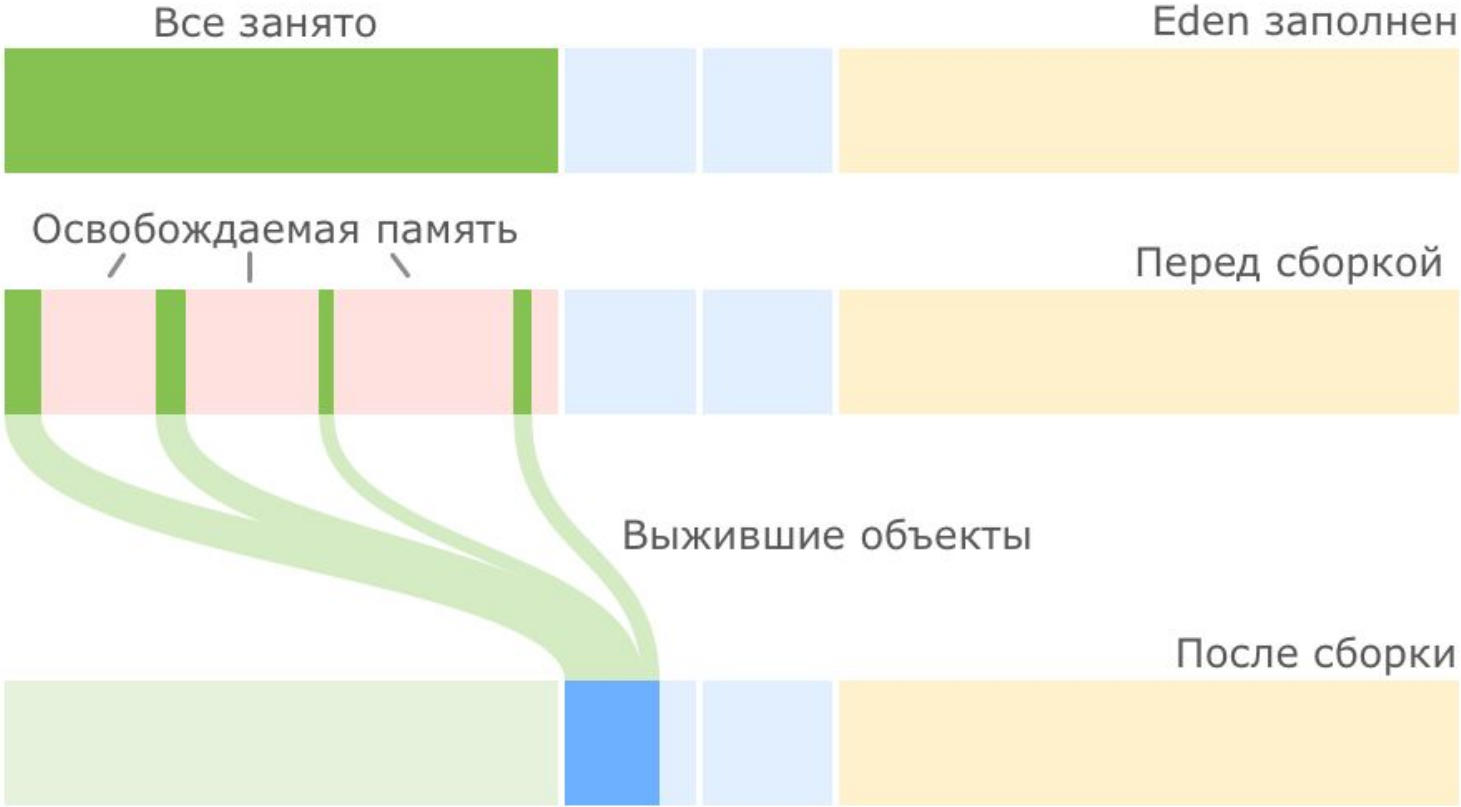
TEXHOATOM

You can control sizes of generations and heap by JVM parameters



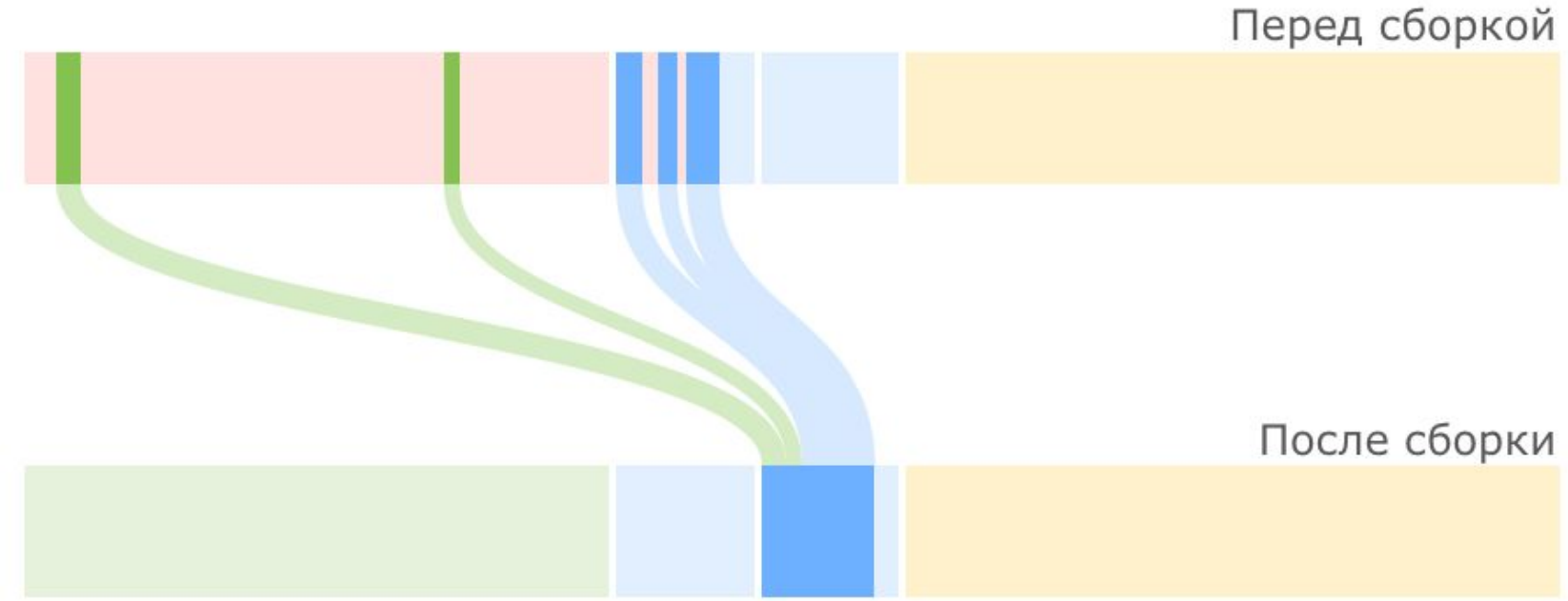
Concurrent Mark Sweep (CMS)

ТЕХНОАТОМ



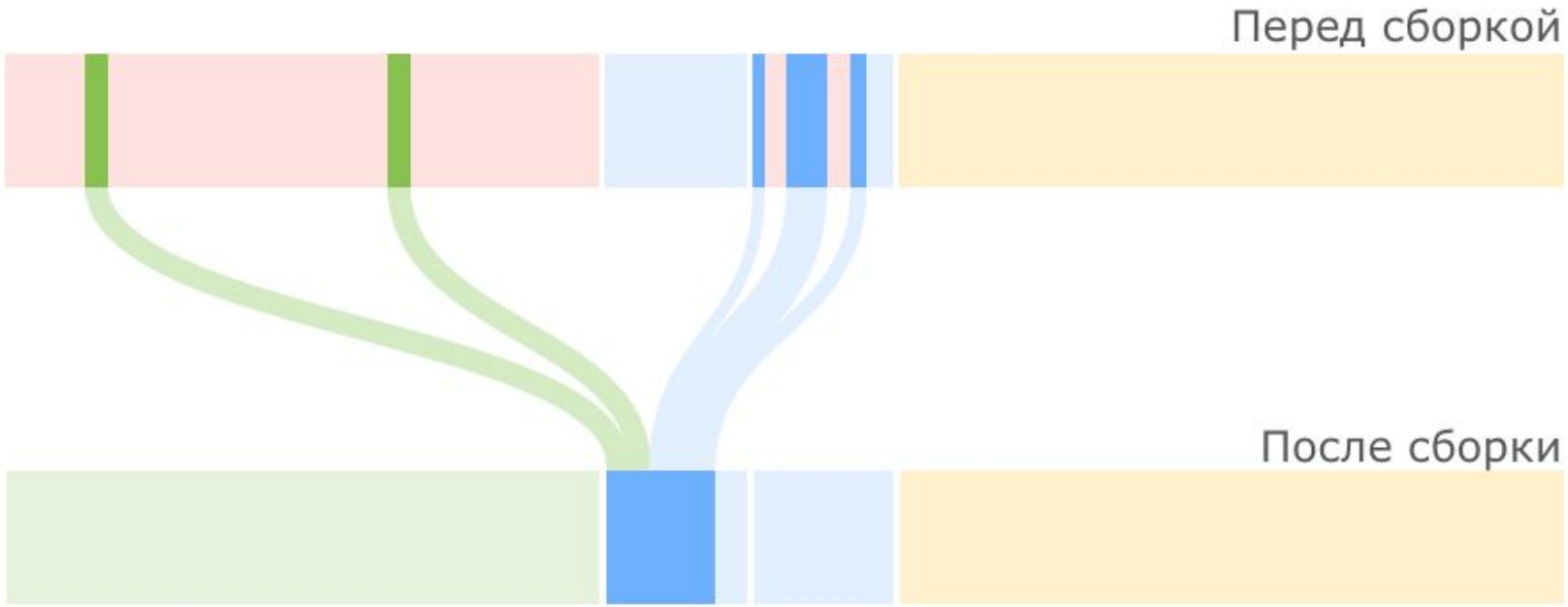
CMS GC to Survival

ТЕХНОАТОМ



CMS Copy Survival

ТЕХНОАТОМ



CMS Copy to Tenured and Compact

ТЕХНОАТОМ

CMS Tenured GC

ТЕХНОАТОМ

GC choosing guidelines

TEXHOATOM

- If the application has a small data set (up to approximately 100 MB), then select the serial collector with the option `-XX:+UseSerialGC`.
- If the application will be run on a single processor and there are no pause time requirements, then let the VM select the collector, or select the serial collector with the option `-XX:+UseSerialGC`.
- If (a) peak application performance is the first priority and (b) there are no pause time requirements or pauses of 1 second or longer are acceptable, then let the VM select the collector, or select the parallel collector with `-XX:+UseParallelGC`.
- If response time is more important than overall throughput and garbage collection pauses must be kept shorter than approximately 1 second, then select the concurrent collector with `-XX:+UseConcMarkSweepGC` or `-XX:+UseG1GC`

Heap sizing guidelines

TEXHOATOM

The following are general guidelines for server applications:

- First decide the **maximum heap size** you can afford to give the virtual machine. Then plot your performance metric against young generation sizes to find the best setting.
  Note that the maximum heap size should always be smaller than the amount of memory installed on the machine to avoid excessive page faults and thrashing.
- If the total heap size is fixed, then increasing the young generation size requires reducing the **tenured generation size**. Keep the tenured generation large enough to hold all the live data used by the application at any given time, plus some amount of slack space (10 to 20% or more).
- Subject to the previously stated constraint on the tenured generation:
  - Grant plenty of memory to the **young generation**.
  - Increase the young generation size as you increase the number of processors, because allocation can be parallelized.

GC Tuning

TEXHOATOM

First we must decide tuning goal, which is specific for application. Common tuning goals are:

- ***Throughput***
  is the percentage of total time not spent in garbage collection considered over long periods of time
- ***Maximum Pause Time***
  are the times when an application appears unresponsive because garbage collection is occurring

https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/toc.html

GC Logging

TEXHOATOM

-Xloggc:filename

causes information about the heap and garbage collection to be writed at each collection to file

```
[GC 325407K->83000K(776768K), 0.2300771 secs]
[Full GC 267628K->83769K(776768K), 1.8479984 secs]
```

-XX:+PrintGCDetails

using the serial garbage collector is shown here

```
[GC [DefNew: 64575K->959K(64576K), 0.0457646 secs] 196016K->133633K(261184K), 0.0459067 secs]
```

-XX:+PrintGCTimeStamps

adds a time stamp at the start of each collection

```
111.042: [GC 111.042: [DefNew: 8128K->8128K(8128K), 0.0000505 secs]111.042: [Tenured: 18154K->2311K(24576K),
0.1290354 secs] 26282K->2311K(32704K), 0.1293306 secs]
```

Essential JVM parameters

TEXHOATOM

The following options are essential. Whenever you crash with Out Of Memory Error, you have a Heap Dump:

```
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=<path to dump>`date`.hprof
-server
```

How to get heap dump

TEXHOATOM

```
jmap -dump:format=b,file=dump.bin <pid>
```

How to analyse heap dump

TEXHOATOM

jhat (included in JDK)

usage. Run:

```
jhat dump.bin
```

then go to localhost:7000 in browser

enjoy! (not really)

MAT (Eclipse Memory Analyzer Tool)

better visual tool

http://www.eclipse.org/mat/

downloads:

http://www.eclipse.org/mat/downloads.php

Profiling

TEXHOATOM

JMC (Java Mission Control) - included in JDK
http://www.oracle.com/technetwork/java/javaseproducts/mission-control/java-mission-control-1998576.html

opt to connect at runtime
or collect profile configured by JVM parameters

To enable profiling:

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder
```

To run automatic profile:

```
-XX:StartFlightRecording=delay=20s,duration=60s,name=MyRecording,filename=myrecording.jfr,settings=profile
```

https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr004.html

One must not trust JMC. It lies in details!

But overall picture is quite accurate

Example

TEXHOATOM

```
@see heap_analysis
```

ПЕРЕРЫВ

Agenda

TEXHOATOM

- Heap, Object layout
- Garbage Collectors
- Performance
- Interview questions

Where is my performance?

TEXHOATOM

Treat performance when it is hurts. Do not optimize prematurely!

Where performance problems mostly come from:

1. Inefficient algorithms and data structures
2. Slow IO
3. GC Problems (memory leaks, too much allocations)
4. everything else

But what if we care about everything else?

How to measure performance?

Whis is tricky thing!

How to measure performance?

TEXHOATOM

Right performance analysis is hard and requires abstraction from numerous factors, that affect performance measurement.
Use the right tool, that bypasses all the tricky corners:

jmh
Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM.
http://openjdk.java.net/projects/code-tools/jmh/

```
@see jmh
```

Agenda

TEXHOATOM

- Heap, Object layout
- Garbage Collectors
- Performance
- Interview questions

Общие вопросы

ТЕХНОАТОМ

1. какие есть в java базовые типы и каков их размер?
2. как в java передаются значения в функцию?
3. чем отличаются interface, class и abstract class
4. как создать объект в java?
5. какие существуют модификаторы доступа у полей/методов/конструкторов
6. что означает ключевое слово static
7. какие есть методы у класса Object?
8. Для чего каждый из них нужен?
9. в чем разница сравнения по == и equals()
10. Контракт eqals()/hashcode()
11. public static void main(String ... args) - почему именно такая сигнатура

Коллекции

ТЕХНОАТОМ

1. Нарисуйте иерархию классов коллекций
2. ArrayList - устройство и асимптотика
3. LinkedList - устройство и асимптотика
4. HashMap - устройство и асимптотика
5. какие требования предъявляются к объектам, помещаемым в hashmap
6. какие требования предъявляются к объектам, помещаемым в treemap
7. Какой размер ArrayList из 10 элементов?

Исключения

ТЕХНОАТОМ

1. Нарисуйте иерархию исключений
2. checked и unchecked exceptions
3. что будет с исключением, выкинутым из блока finally?
4. что такое try-with-resources?
5. что будет с исключением, выкинутым при закрытии ресурса?

Многопоточность

ТЕХНОАТОМ

1. Как создать поток в java?
2. как остановить поток в java?
3. ключевое слово final
4. ключевое слово volatile
5. ключевое слово synchronized
6. как понять, что случился deadlock?

Остальное

ТЕХНОАТОМ

1. Строки
    a. как сравнивать строки?
    b. расскажите про интернирование строк
2. Сериализация
    a. как сериализовать и десеарилизовать объект в java?
    b. ключевое слово transient
3. GC
    a. как увеличить размер кучи в java?
    b. как работает GC в java?

Знаете ли Вы?

ТЕХНОАТОМ

1. знаете ли Вы sql?
2. знаете ли Вы hibernate?
3. знаете ли Вы Spring?

References

TEXHOATOM

Conferences:

https://www.oracle.com/javaone/index.html

http://jpoint.ru/

http://jug.ru/

Community:

http://razbor-poletov.com/

https://gitter.im/razbor-poletov/razbor-poletov.github.com

Books:

https://www.amazon.com/Thinking-Java-4th-Bruce-Eckel/dp/0131872486 (basic)

https://www.amazon.com/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601 (deep concurrency)

Other:

https://shipilev.net/


КОНЕЦ

ТЕХНОАТОМ

Спасибо за внимание!

Александр Помосов

alpieex@gmail.com