概要：

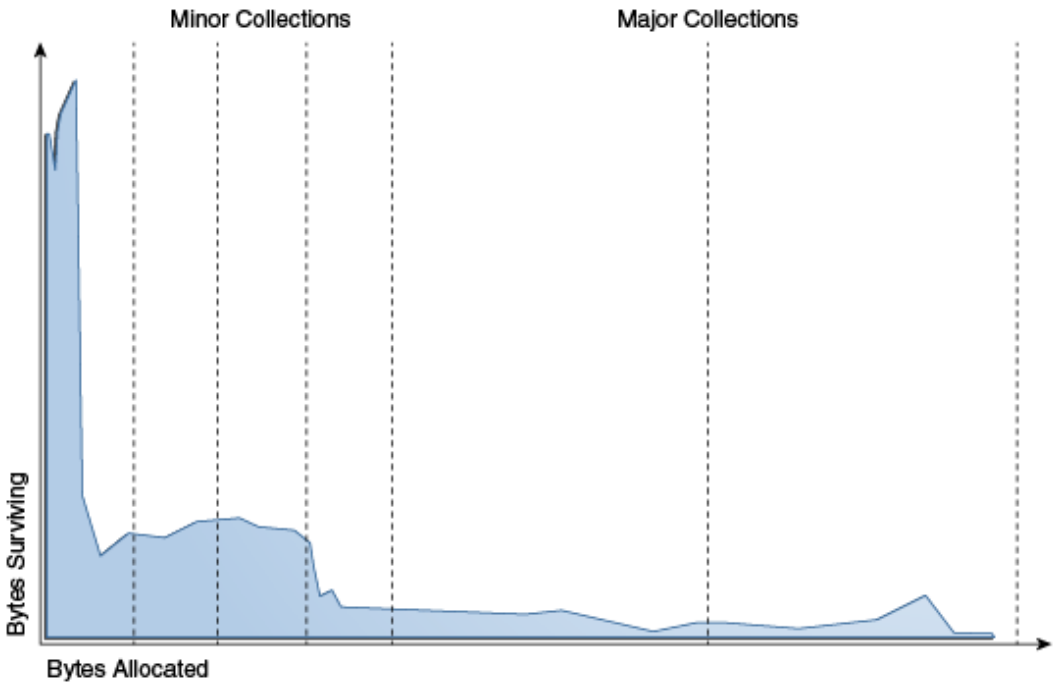
- **目的**：简单了解 jvm gc 原理，对 jvm gc 调优有大概的认识，问题排查的工具方法
- 内容：
  - jvm 内存模型、几种 gc collector
  - 简单 gc 调优和配置
  - jdk 内置工具，监视和分析
- 其他：主要基于 jdk8 官方文档

一、 GC 机制

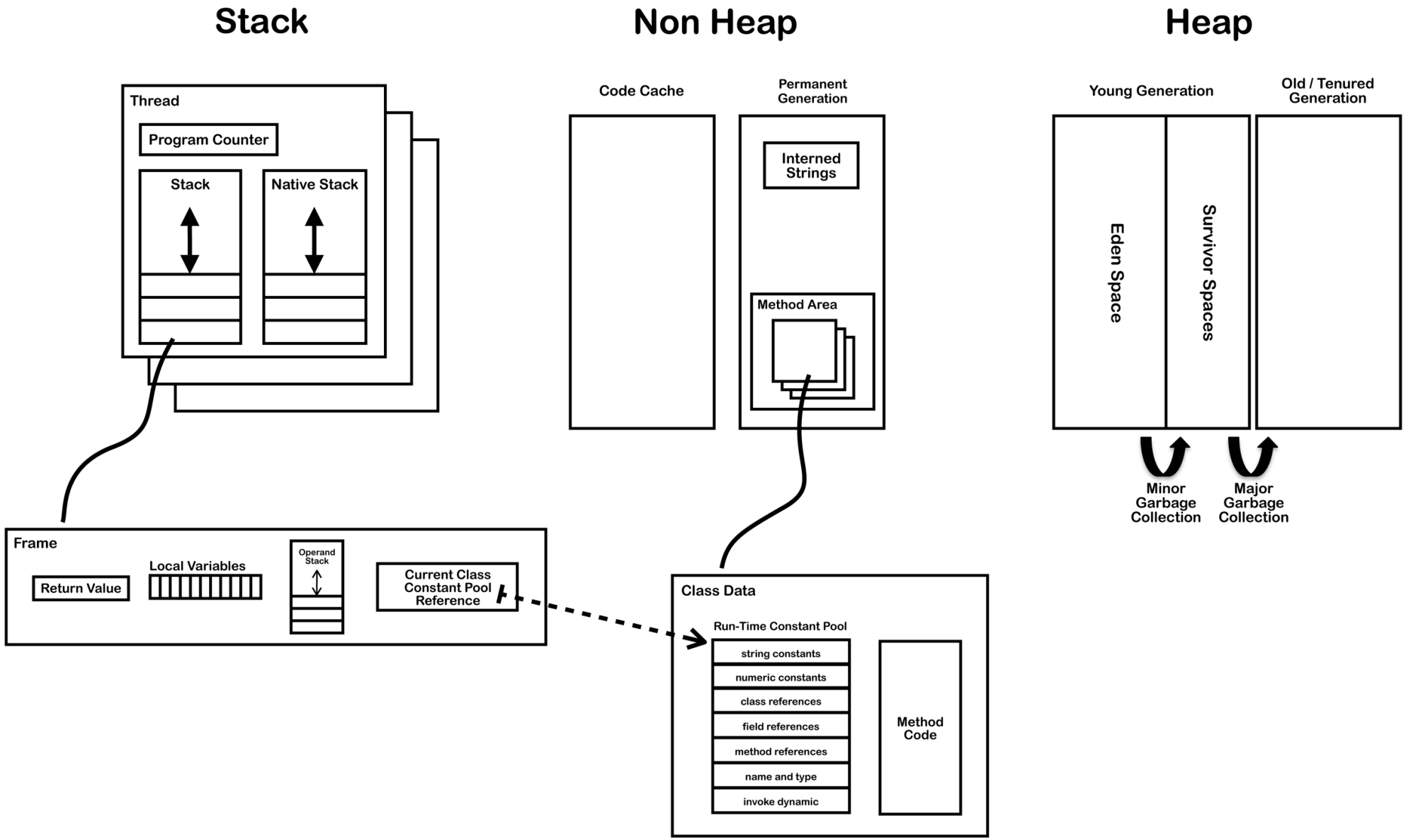
对象的短生命周期

- 大部分对象“死得早”
- 新生对象很少引用生存时间长的对象

- web 应用中的典型的场景：
  - 请求处理过程中 new 的大部分对象很快 unreachable（占绝大部分
  - 处理时间较长的方法运行中创建的对象（可能活过几次 minor gc
  - 长时间对象（可能很长，但早晚还是会被销毁
  - 长生命周期对象（常量、固定内存数据



内存模型（老版

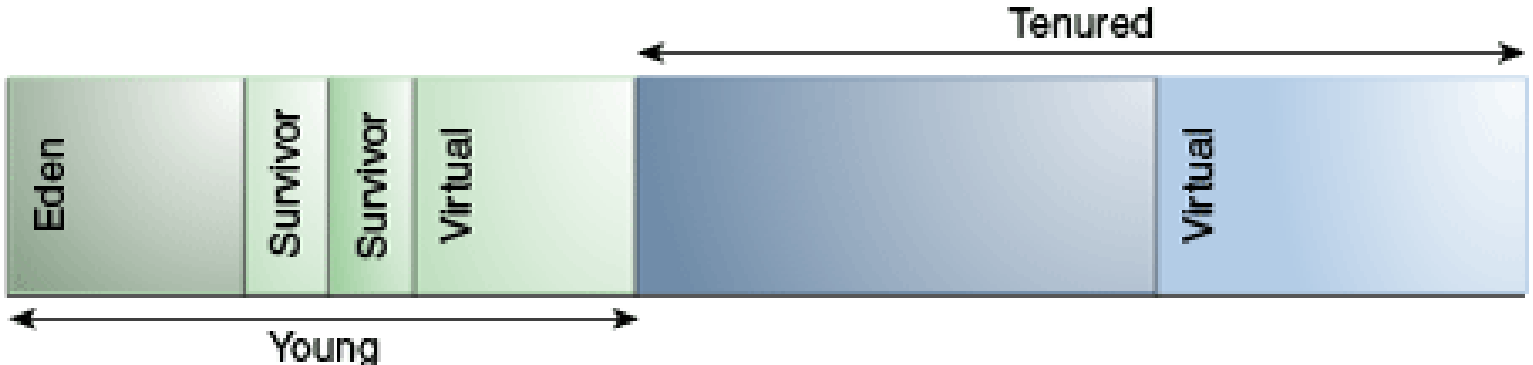

Stack
Non Heap
Heap
Thread
Program Counter
Stack
Native Stack
Code Cache
Permanent Generation
Interned Strings
Method Area
Young Generation
Old / Tenured Generation
Eden Space
Survivor Spaces
Minor Garbage Collection
Major Garbage Collection
Frame
Return Value
Local Variables
Operand Stack
Current Class Constant Pool Reference
Class Data
Run-Time Constant Pool
string constants
numeric constants
class references
field references
method references
name and type
invoke dynamic
Method Code

GC 基本规则： stop-the-world

- : Stop-the-world 会在任何一种 GC 算法中发生（不一定是全程）。Stop-the-world 意味着 JVM 因为要执行 GC 而停止了应用程序的执行。当 Stop-the-world 发生时，除了 GC 所需的线程以外，所有线程都处于等待状态，直到 GC 任务"完成"。GC 优化很多时候就是指减少 Stop-the-world 发生的时间（最长中断时间、中断时间比

基于年代的对象堆内存管理

- 新生代(young generation： minor gc(YGC)copying collector.
  - eden: 新对象首先被分配到该区域（如果对象很大则直接分配到老年代
  - 存活区(survivor)1 2： minor gc 后存活的对象被交替移动到其中一个区域，在该区域存活一定次数后移动到老年代（如果 minor gc 时该区域不足也会被直接移动到老年代
- 老年代(old generation: major gc(FGC)mark-sweep-compact collection
- 持久代(perm: 类元信息、常量、字符串等， FGC 时同时回收改区域（ JDK8 取消了该区域， MetaData ）



可能的问题

- java.lang.VirtualMachineError
  - OutOfMemory
  - StackOverFlow

- gc 停顿（ StopTheWorld，甚至响应超时
- 频繁 gc，吞吐量达不到要求

衡量指标

- 吞吐量 (Throughput): 处理业务所用时间（除去 gc 时间）占比
- 暂停时间：由于 gc 而暂停响应的时间

五种 gc 算法

1.Serial GC： mark-sweep-compact，单线程，高效，适合小内存程序或者单核服务器

- -XX:UseSerialGC

- 2.Parallel GC(Throughput collector)： 多线程，适合多核服务器，内存占用中一大
- 3.Parallel Old GC (Parallel Compacting GC)： 区别于老年代使用单线程 gc

- 》》 Throughput collector,thread count

- 》》 Concurrent collector
- 4.Concurrent Mark& Sweep GC （or “CMS”）: 优先响应时间，弱化吞吐，以 cpu 资源换取 shorter major collection pause time （a large tenured && 2+ cpu），gc 过程中并非所有阶段都 stop-the-world ,(Scheduling Pauses)
  - 缺点： cpu 内存消耗增加，当碎片过多需要压缩时， stop-the-world 时间更长，相对的 cpu 少时效果不是很理想（增量模式， 1 or 2cpu， @Deprecated
- 5.Garbage First (G1) GC ，最快的，针对高配服务器，算法想对复杂，调优也比较难搞， jdk8 推荐

When is a garbage collection started?

System.gc()?

默认值：

- 2cpu && >=2G ： server-class machine

>>>

Throughput garbage collector， Initial heap size 1/64x ～ 1G， Maximum heap size ¼ ～ 1G， Server runtime compiler ： ： 64bit parallel collector

4. java -XX:+PrintFlagsFinal <GC options> -version| grep MaxHeapSize
5. +PrintFlagsInitial
6. -XX:+PrintCommandLineFlags

gc 算法的一般选择步骤

Unless your application has rather strict pause time requirements, first run your application and allow the VM to select a collector. If necessary, adjust the heap size to improve performance. If the performance still does not meet your goals, then use the following guidelines as a starting point for selecting a collector.

- 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.
-

二、 gc 调优

基于人体工程学的 jvm 自动调优

- 一、最短暂停时间
  - -XX:MaxGCPauseMillis=<nnn>
- 二、吞吐量优先
  - -XX:GCTimeRatio=<nnn> ， The ratio of garbage collection time to application time is 1 / (1 + <nnn>)

footprint ： － Xmx ／＋ maximum heap size

》》底层参数

参数确定一般步骤

- 不能因为某个应用使用的 GC 参数“ A”，就说明同样的参数也能给其他服务带来最佳的效果。而是要因地制宜，有的放矢。
- gc 参数的也是需要变化的：服务器配置变更、业务代码改动、期望效果的提升、流量变化。。。

1. 确定基准值和期望
    1. maximum heap size (<physicmemory， pause time， throughput
2. 调整细节参数以满足要求
3. 发生变化时调整参数（资源、流量、业务、期望值
4. 总之，大概就是观察一 > 调整 直到合适

参数配置

- Java reference：
  - http://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html
- Java -X
- HotSpot VM
  - http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html

- Tomcat,readme

参数

- gc 算法
- 堆内存空间 -Xms4G–XX:MaxHeapSize -XX:InitialHeapSize=6m -XX:InitialSurvivorRatio=ratio
- 新生代空间 -Xmn512m–XX:NewSize–XX:MaxNewSize –XX:NewRatio=2 –XX:NewSize 。。。
- Perm

- Gc 算法，各算法的特殊参数（线程数量、开关、内存页、空闲百分百。。。
- Advanced Garbage Collection Options （各 gc 文档、默认值列表

- Tomcat 配置

GC 监控

- Jstat
- (Java) VisualVM + Visual GC, jconsole
- 第三方

- **哨兵系统**
- Gc log **分析**

- 前提：
  - 已经通过 -Xms 和 - Xmx 设置了内存大小
  - 包含了 -server 参数
- 系统中没有超时日志等错误日志

优化之前

- Gc 优化是最后一部

- 最小变量范围
- 减少不必要的对象生成
- StringBuilder StringBuffer 替换 String
- 减少日志输出

目的

- 一个是将转移到老年代的对象数量降到最少
- 另一个是减少 Full GC 的执行时间

优化的过程

- 1. 监控 GC 状态
- 2. 在分析监控结果后，决定是否进行 GC 优化
- 3. 调整 GC 类型 / 内存空间
- 4. 分析结果
- 5. 如果结果令人满意，你可以将该参数应用于所有的服务器，并停止 GC 优化
-
- 结合程序特性分析：流量波动　业务对象特性　算法特性　业务架构特点

其他，架构参数

- Nginx
- apache

三、工具和问题排查

OutOfMemoryError

- https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html
- -XX: PermSize" and "-XX: MaxPermSize" ()
- Tomcat 多应用部署，重启时 class 无法卸载完全
- -Xmx

……

- -XX:+HeapDumpOnOutOfMemoryError and -XX:HeapDumpPath

- *Java heap space: -Xmx ; app is holding refs to objects unintentionally ; finalize()...
- *GC Overhead limit exceeded: gc time>98%
- Requested array size exceeds VM limit:
- Metaspace:
- request size bytes for reason. Out of swap space?:
- Compressed class space:
- reason stack_trace_with_native_method:

tools

- jconsole, Jvisualvm,Jcmd(8, Jmc(8
- Jps, Jstat, Jstatd
- Jinfo,jhat,jmap,jsadebugd,jstack

- Gc analysis

jps

- jps -m -l -v

jinfo

- Generates configuration information.
- http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jinfo.html

jconsole/jvisualvm/Java mission control

- ref:(通常可以用于本地或测试环境分析调试
- http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jconsole.html
- http://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html
- tomcat jconsole
- https://tomcat.apache.org/tomcat-7.0-doc/monitoring.html

Jstat/gc log

- jstat -options
- http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jstat.html
- （参数和现实意义

jhat

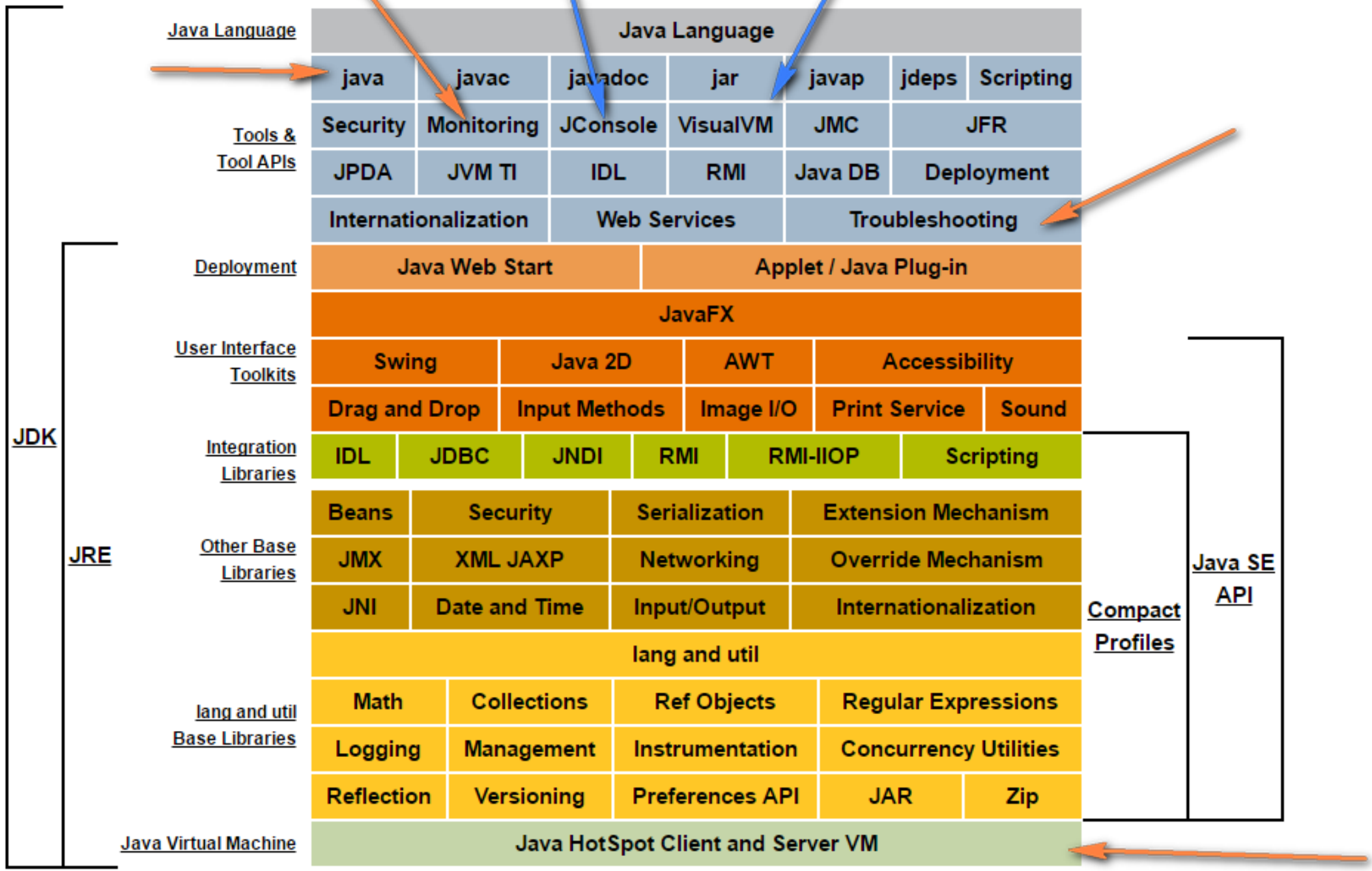
- -J-Xmx4g

jmap

- http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jmap.html
- jmap -J-d64 -heap pid.
- -dump:file=dp.hsdp

参考

- Jdk 官方文档
- http://docs.oracle.com/javase/6/docs/
- http://docs.oracle.com/javase/7/docs/
- http://docs.oracle.com/javase/8/docs/
- http://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html
- Java -help | java-? | java -X


Java Language
Java Language
java
javac
javadoc
jar
javap
jdeps
Scripting
Tools & Tool APIs
Security
Monitoring
JConsole
VisualVM
JMC
JFR
JPDA
JVM TI
IDL
RMI
Java DB
Deployment
Internationalization
Web Services
Troubleshooting
Deployment
Java Web Start
Applet / Java Plug-in
JavaFX
User Interface Toolkits
Swing
Java 2D
AWT
Accessibility
Drag and Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC
JNDI
RMI
RMI-IIOP
Scripting
Other Base Libraries
Beans
Security
Serialization
Extension Mechanism
JMX
XML JAXP
Networking
Override Mechanism
JNI
Date and Time
Input/Output
Internationalization
lang and util
lang and util Base Libraries
Math
Collections
Ref Objects
Regular Expressions
Logging
Management
Instrumentation
Concurrency Utilities
Reflection
Versioning
Preferences API
JAR
Zip
Java Virtual Machine
Java HotSpot Client and Server VM
JDK
JRE
Compact Profiles
Java SE API

- jdk7 HotSportOptions：
  http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html
- jdk67 GC 调优：
  http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
- jdk8 GC 调优：
  http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/toc.html
- gc 诊断（ gc log ）：
  http://www.oracle.com/technetwork/java/example-141412.html
- HotSpot GC FAQ： http://www.oracle.com/technetwork/java/faq-140837.html
- HotSpot JVM FAQ：
  http://www.oracle.com/technetwork/java/hotspotfaq-138619.html

http://www.importnew.com/1993.html
http://www.importnew.com/2057.html
http://www.importnew.com/3146.html
http://www.importnew.com/3151.html
http://www.importnew.com/13954.html
(建议看原文，翻译有些地方不太准确

https://blog.codecentric.de/en/2012/07/useful-jvm-flags-part-1-jvm-types-and-compiler-modes/
https://blog.codecentric.de/en/2012/07/useful-jvm-flags-part-2-flag-categories-and-jit-compiler-diagnostics
https://blog.codecentric.de/en/2012/07/useful-jvm-flags-part-3-printing-all-xx-flags-and-their-values/
https://blog.codecentric.de/en/2012/07/useful-jvm-flags-part-4-heap-tuning/
https://blog.codecentric.de/en/2012/08/useful-jvm-flags-part-5-young-generation-garbage-collection/

提醒：

- 只提供大概的印象，如果需要实干，务必过一遍文档
- 建议官方文档，并且一致的版本
- 调优和问题排查都是长期的过程，多观察

完！