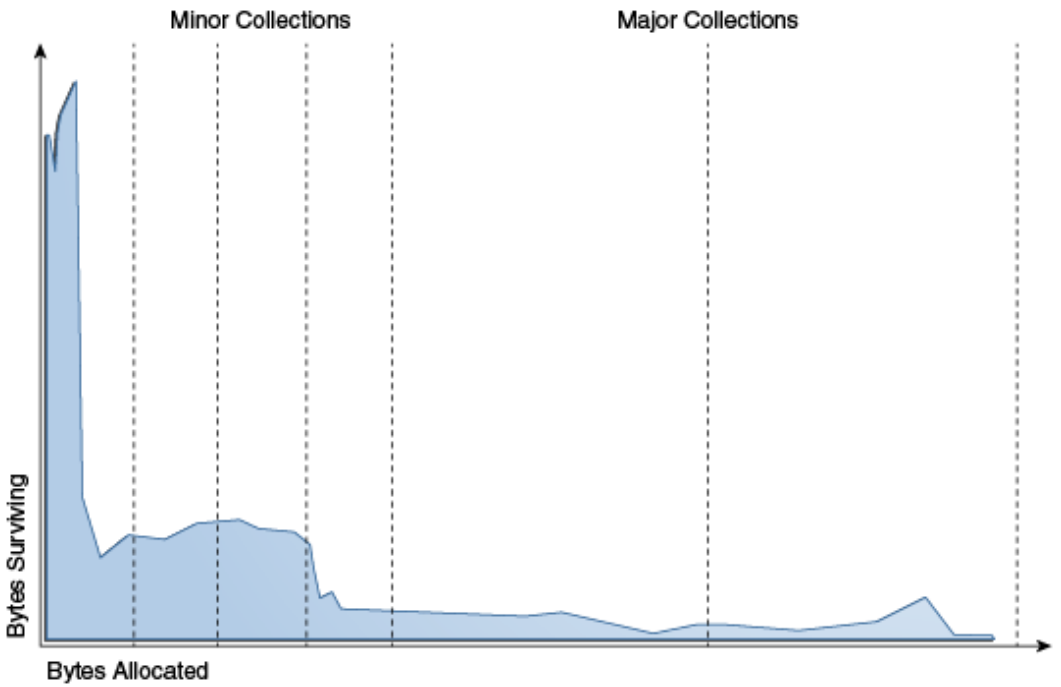
一、GC机制和调优简介

概要：

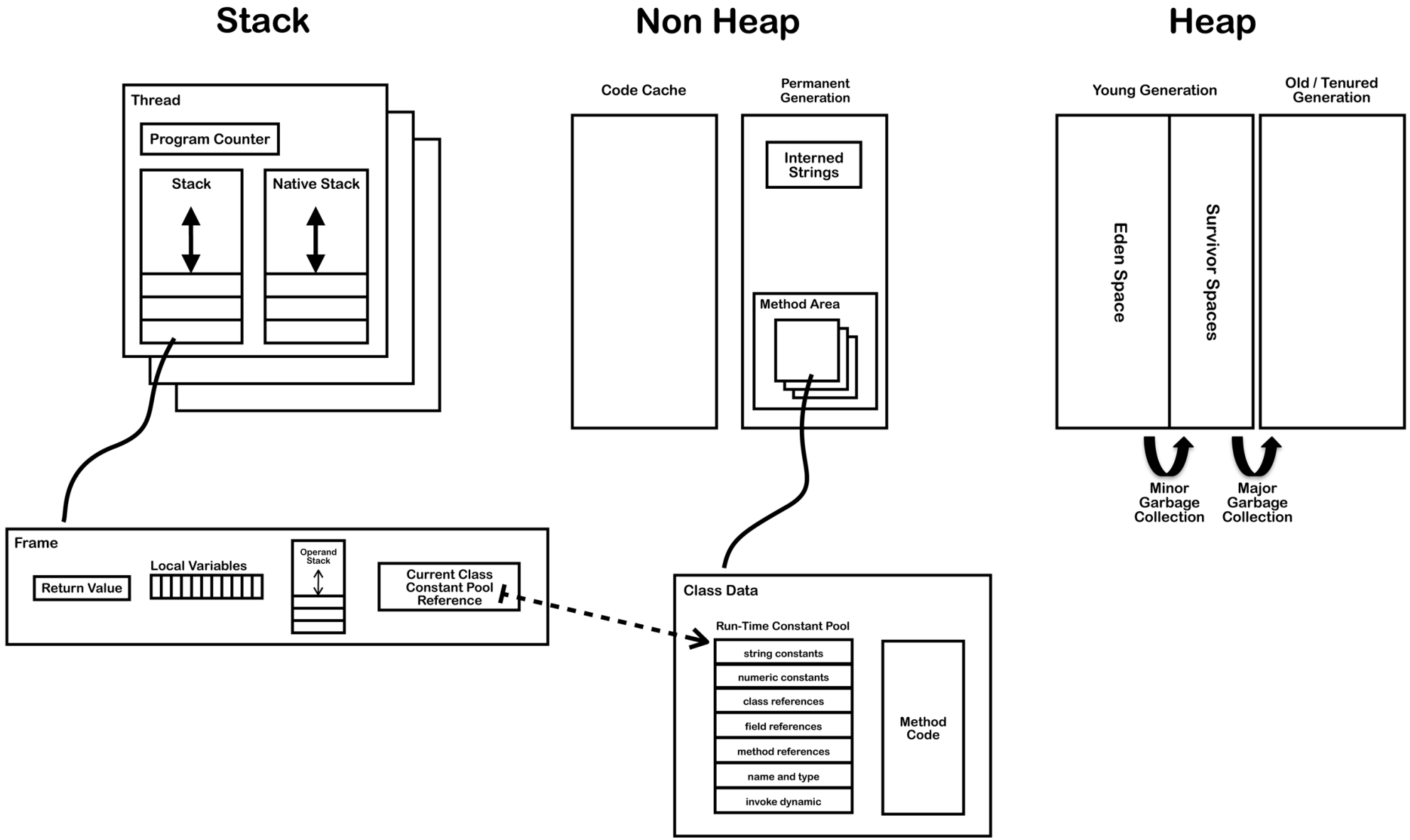
- **目的**：简单了解jvm gc原理，对jvm gc调优有大概的认识
- 内容：
  - jvm内存模型、几种gc collector
  - 简单gc调优和配置
- 其他：基于jdk8文档

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（定时归纳评分举例,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

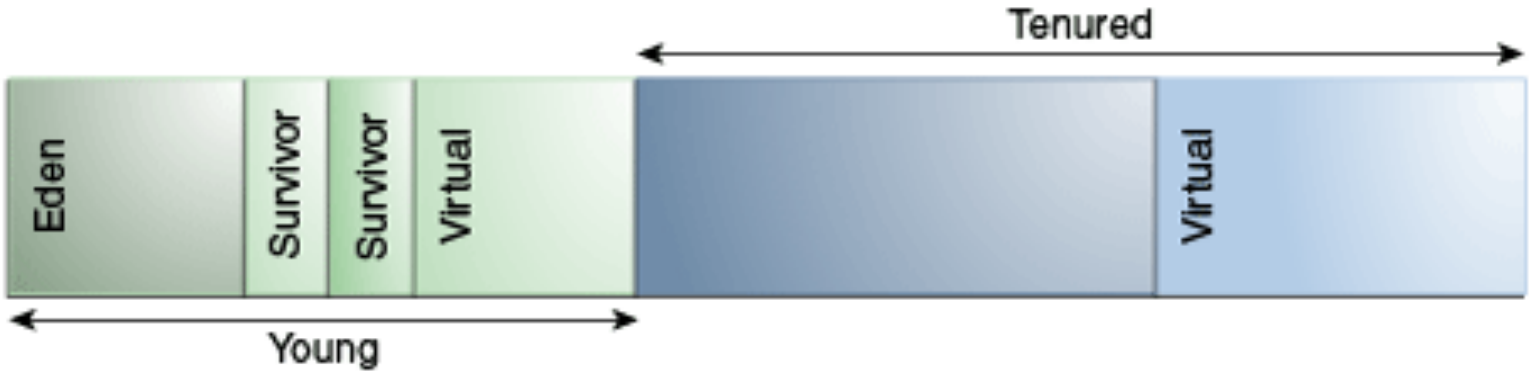

Stack
Non Heap
Heap
Thread
Program Counter
Stack
Native Stack
Code Cache
Permanent Generation
Interned Strings
Method Area
Young Generation
Old / Tenured Generation
Eden Space
Survivor Spaces
Minor Garbage Collection
Major Garbage Collection
Frame
Return Value
Local Variables
Operand Stack
Current Class 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

基于年代的对象内存管理

- 新生代(young generation：minor gc(YGC)
  - 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)
- 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

- gc停顿（StopTheWorld，甚至响应超时
- OutOfMemory（or 98%gc time && 2% heap recovered
- 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：单线程，高效，适合小内存程序或者单核服务器
  - -XX:UseSerialGC
  - Parallel/ Throughput collector
- 2.Parallel GC(Throughput collector)： 多线程，适合多核服务器，内存占用中一大 （老年代使用单线程
- 3.Parallel Old GC (Parallel Compacting GC)：区别于老年代也执行并行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

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. -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.

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

1. 确定maximum heap size (<physicmemory
2. 增加新时代空间，同时也调高stack space
   - 保证老年代空间满足最大同时存活容量
3. 增加cpu时调大新生代内存

- 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。。。

参数配置

- Java
- Java –X
- Tomcat
- HotSpot VM

参数

- 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
- Advanced Garbage Collection Options
- Tomcat配置

二、GC监控

- CUI GC监控方法使用一个独立的叫做”jstat”的CUI应用，或者在启动JVM的时候选择JVM参数”verbosegc”。
- Jstat
- (Java) VisualVM + Visual GC

- **哨兵系统**
- Gc log**分析**

三、gc优化

- 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

四、架构参数

- Nginx
- apache

OutOfMemoryError

- -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

tools

- Jcmd, Jmc,jconsole, Jvisualvm
- Jps, Jstat, Jstatd
- Jinfo,jhat,jmap,jsadebu gd,jstack
- Gc analysis

参考

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

- 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

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/