ORACLE®

甲骨文

以下内容旨在概述产品的总体发展方向。该内容仅供参考，不可纳入任何合同。该信息不承诺提供任何资料、代码或功能，并且不应该作为制定购买决策的依据。描述的有关 Oracle 产品的任何特性或功能的开发、发行和时间规划均由 Oracle 自行决定。

超强 (WLS/Java) 性能研讨会

JVM 性能调优

Zhao Yi

咨询解决方案架构师 — OFM A 团队

ORACLE

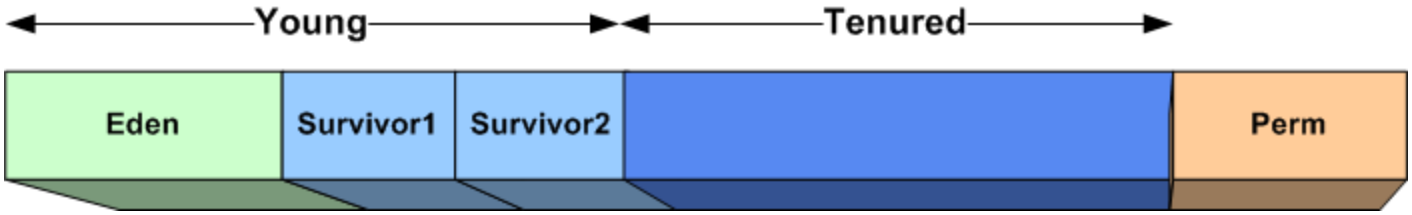
甲骨文

议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

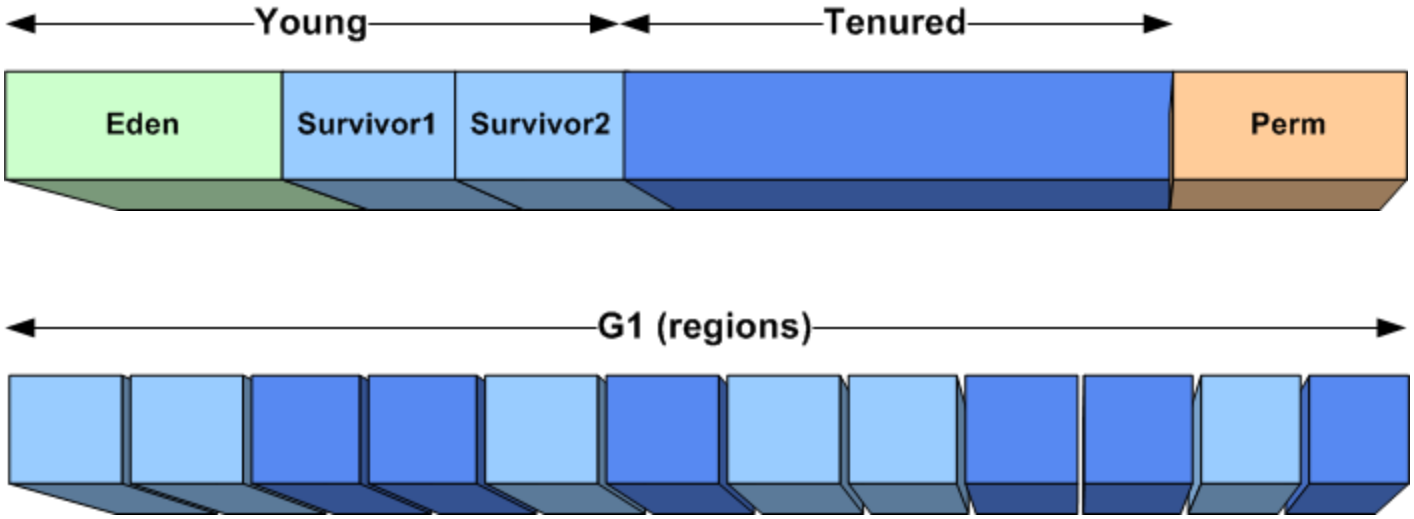
JVM 基础知识

Hotspot 堆 — JDK 6



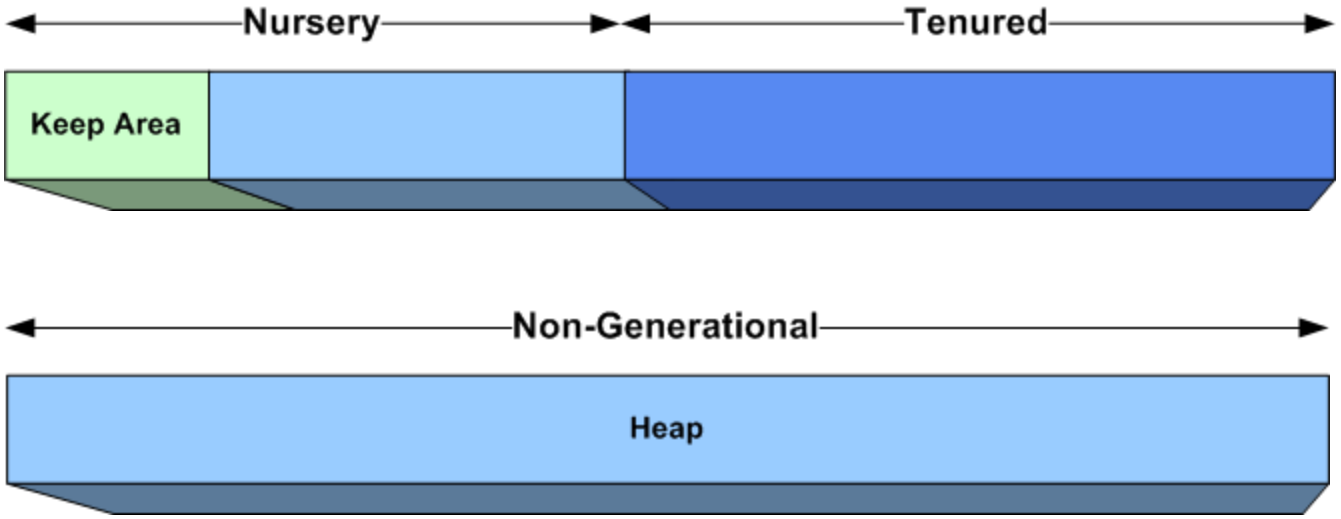
JVM 基础知识

Hotspot 堆 — JDK 7



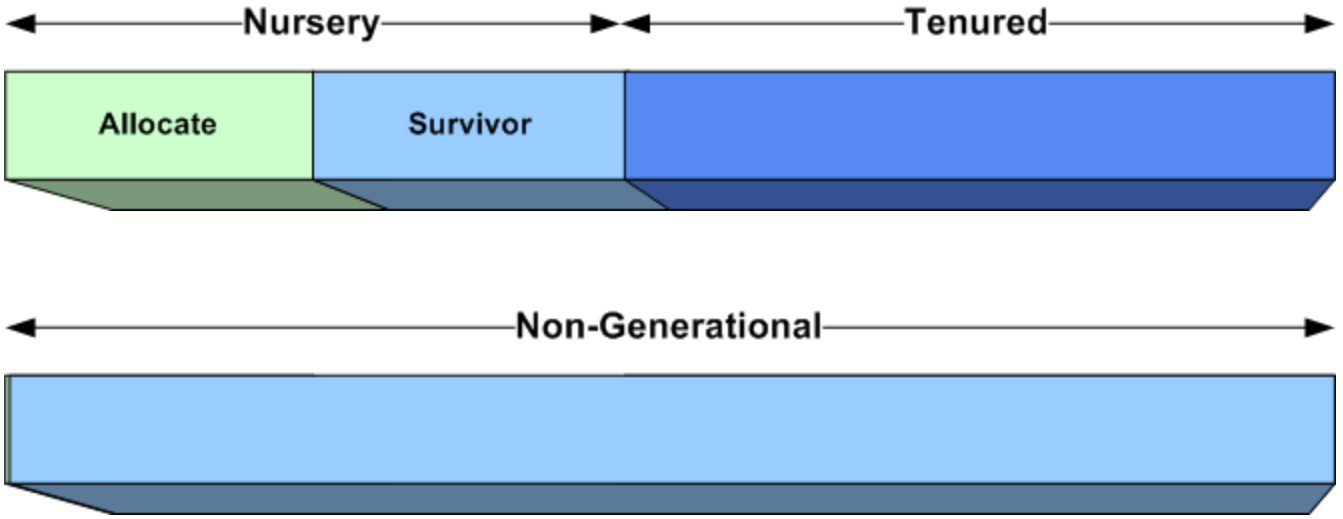
JVM 基础知识

JRockit 堆



JVM 基础知识

IBM JVM



JVM 基础知识

Java 编译

- 解释模式
  - 首先对字节码 (.class) 进行解释*
- 实时 (JIT) 编译
  - 被标记为“热点”时，字节码会编译为本机代码
    - 例如，在 Hotspot JVM 中，方法执行的次数超过“-XX:CompileThreshold”
    - *JRockit 仅在编译模式下运行
- 优化
  - JVM 收集启发式方法来优化编译后的代码

JVM 基础知识

自适应内存管理

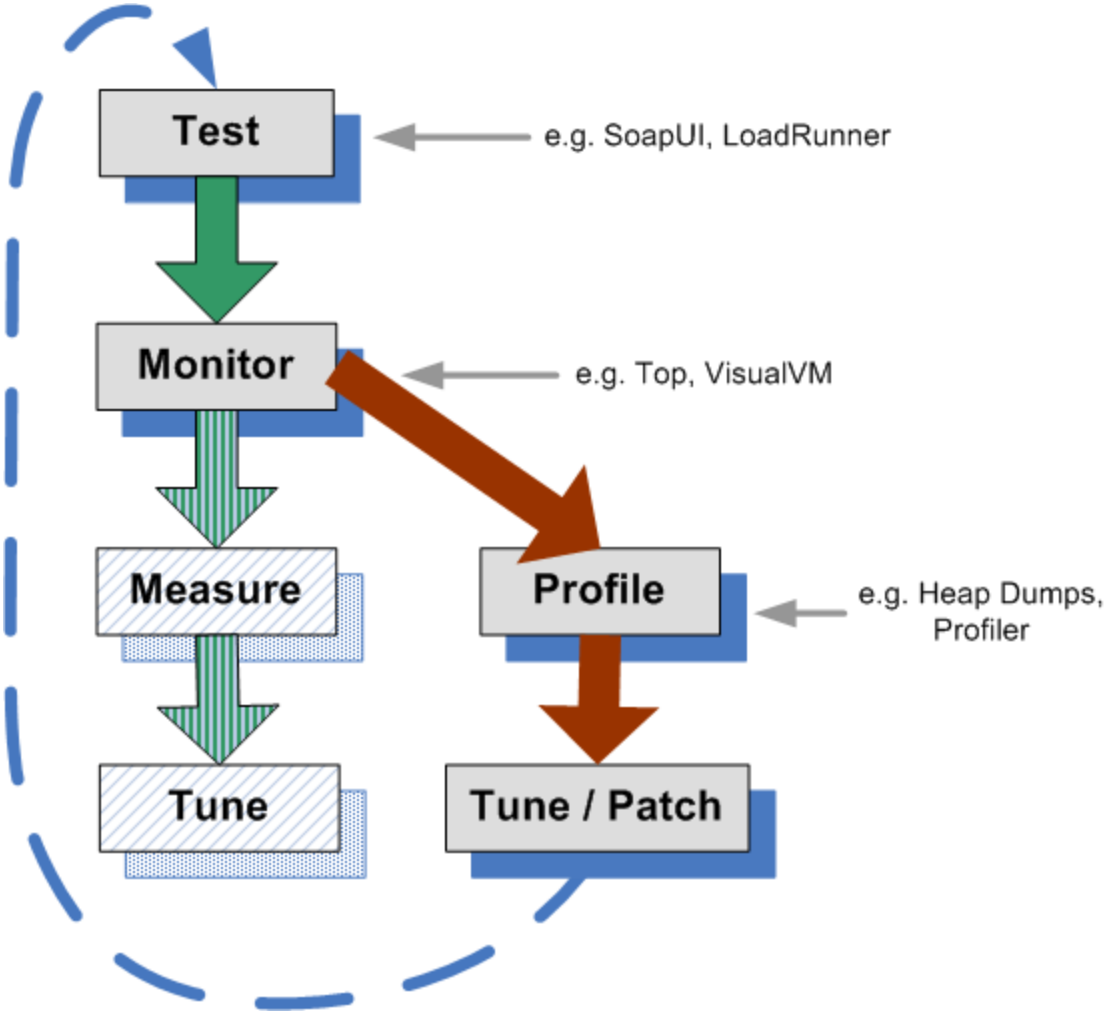
- Hotspot 将其称作“人体工程机制”
- 收集堆使用情况和 GC 统计数据
- 启动时会根据服务器类型选择一些值
  - 例如，代的大小、堆大小、gc 线程
- 一些值会根据统计数据动态调整
  - 例如，存活空间
  - JRockit 会动态切换 GC 算法
- 优良的即用性能
- *精心手动调优的 JVM 可实现更佳的性能*

议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

JVM 性能调优

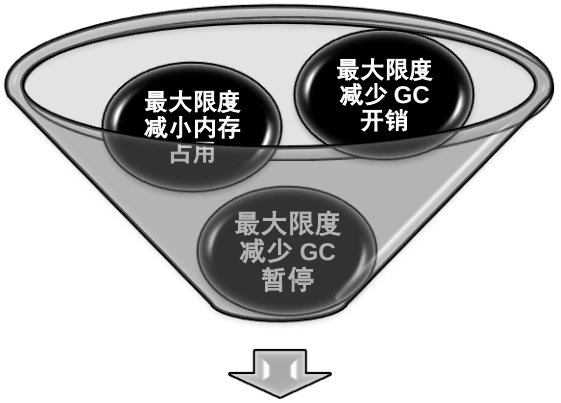
性能调优流程



JVM 性能调优

调优的性能指标

- **内存占用**
  - 在 64 位环境中重要性较低
  - 防止内存交换
- **启动时间**
  - 比较重要（例如生产环境）
- **吞吐量**
  - 一段时间内处理的事务量（例如 TPS）
  - 重要
- **响应速度**
  - 延迟、往返时间、用户等待时间等
  - 非常重要
  - 注意：一个常见的错误是使用单一消息延迟来反映机器性能



您只能选择两项！

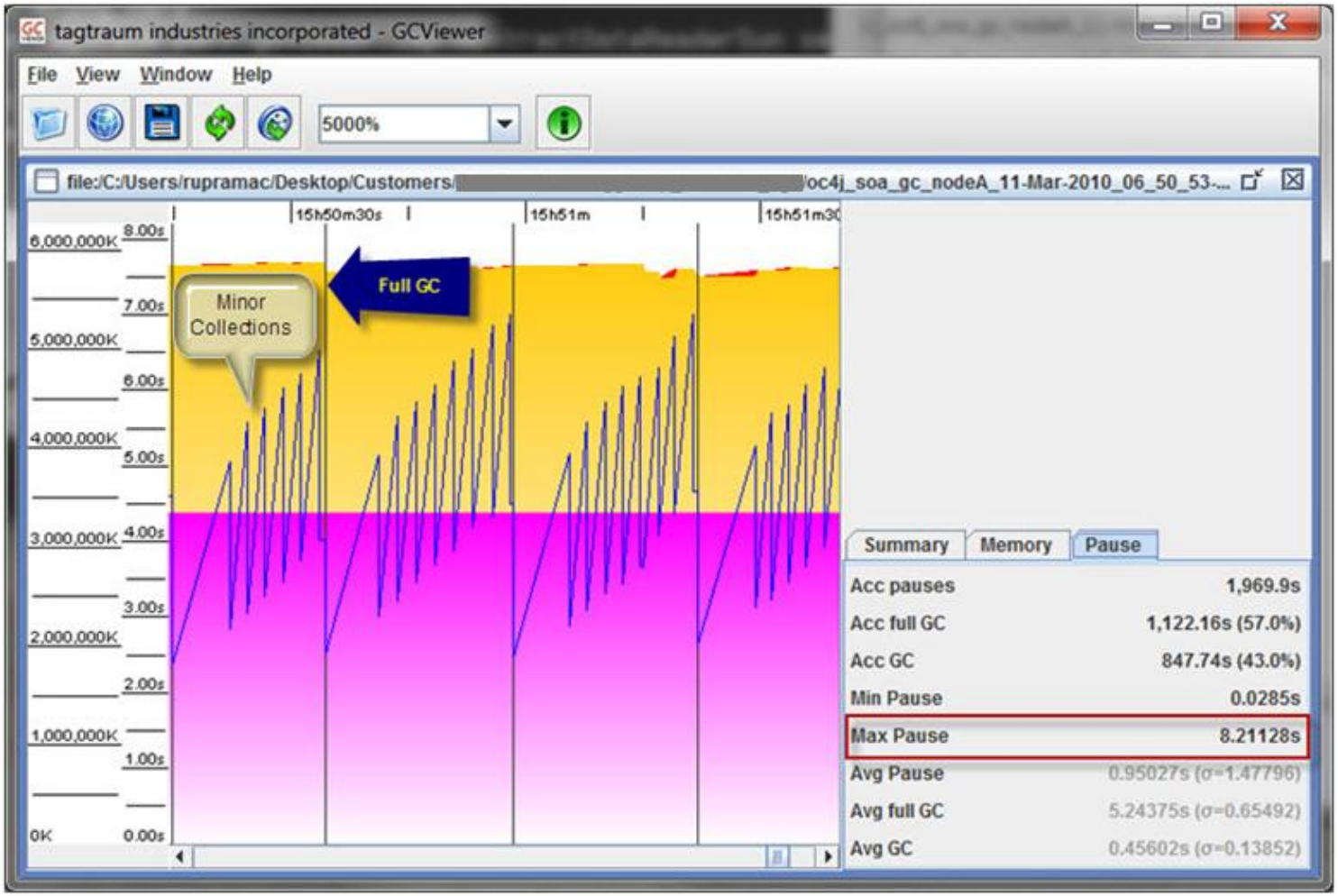
JVM 性能调优

JVM 监视和分析

- JVM 监视工具
  - 联机：JRMC (JRockit)、VisualVM (Hotspot)
  - 脱机：GC 日志、GCViewer（所有）、JFR
- JVM 分析
  - 内存、CPU、锁定/监视分析
  - 分析器
    - 大多数 IDE (JVMTI)
    - MAT（堆转储）
    - JRMC (JRockit)
    - VisualVM (Hotspot)
    - 第三方分析器（JProfiler、YourKit）

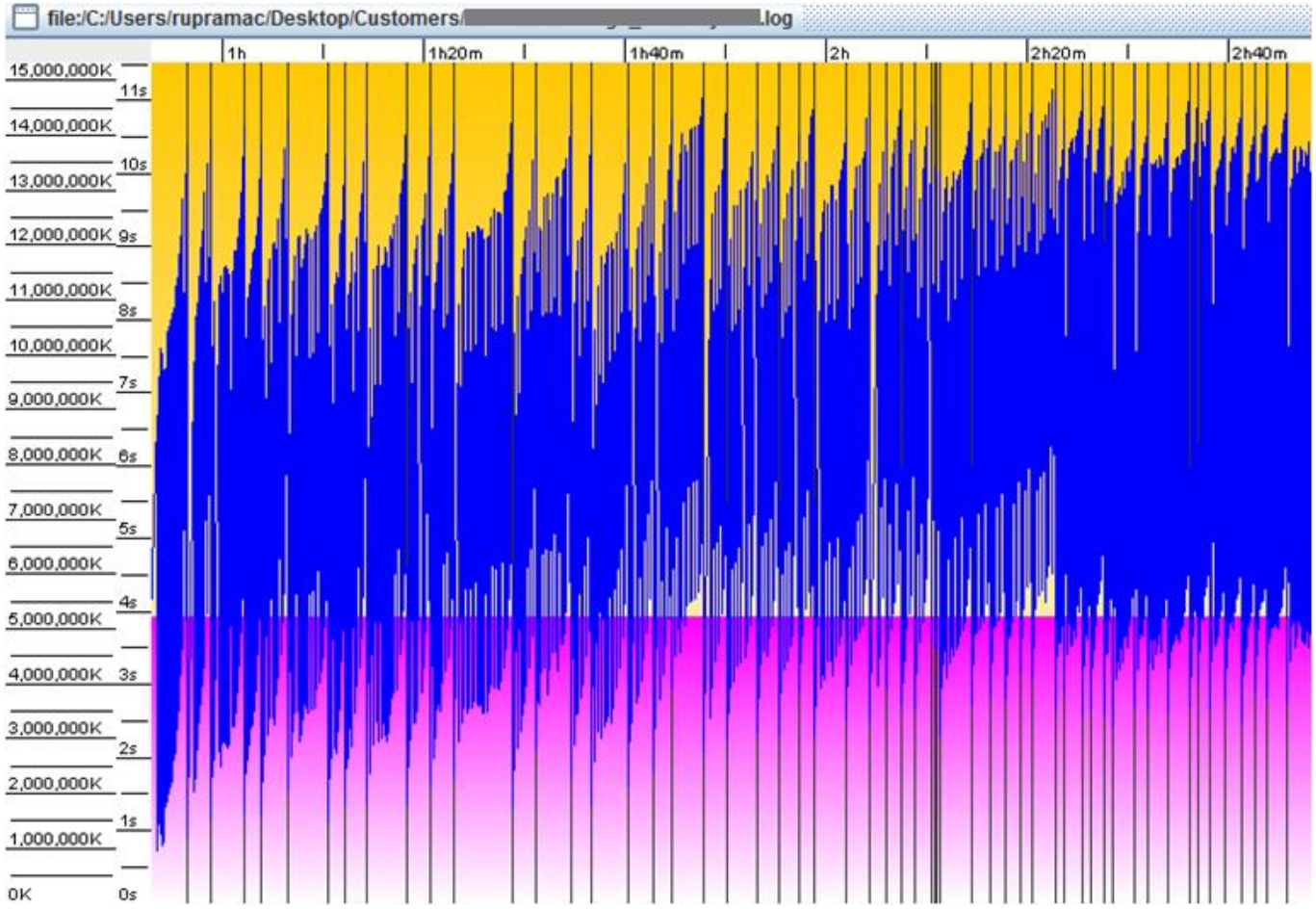
JVM 性能调优

GCViewer — 脱机分析 GC 日志



JVM 性能调优

GCViewer — 内存泄漏模式

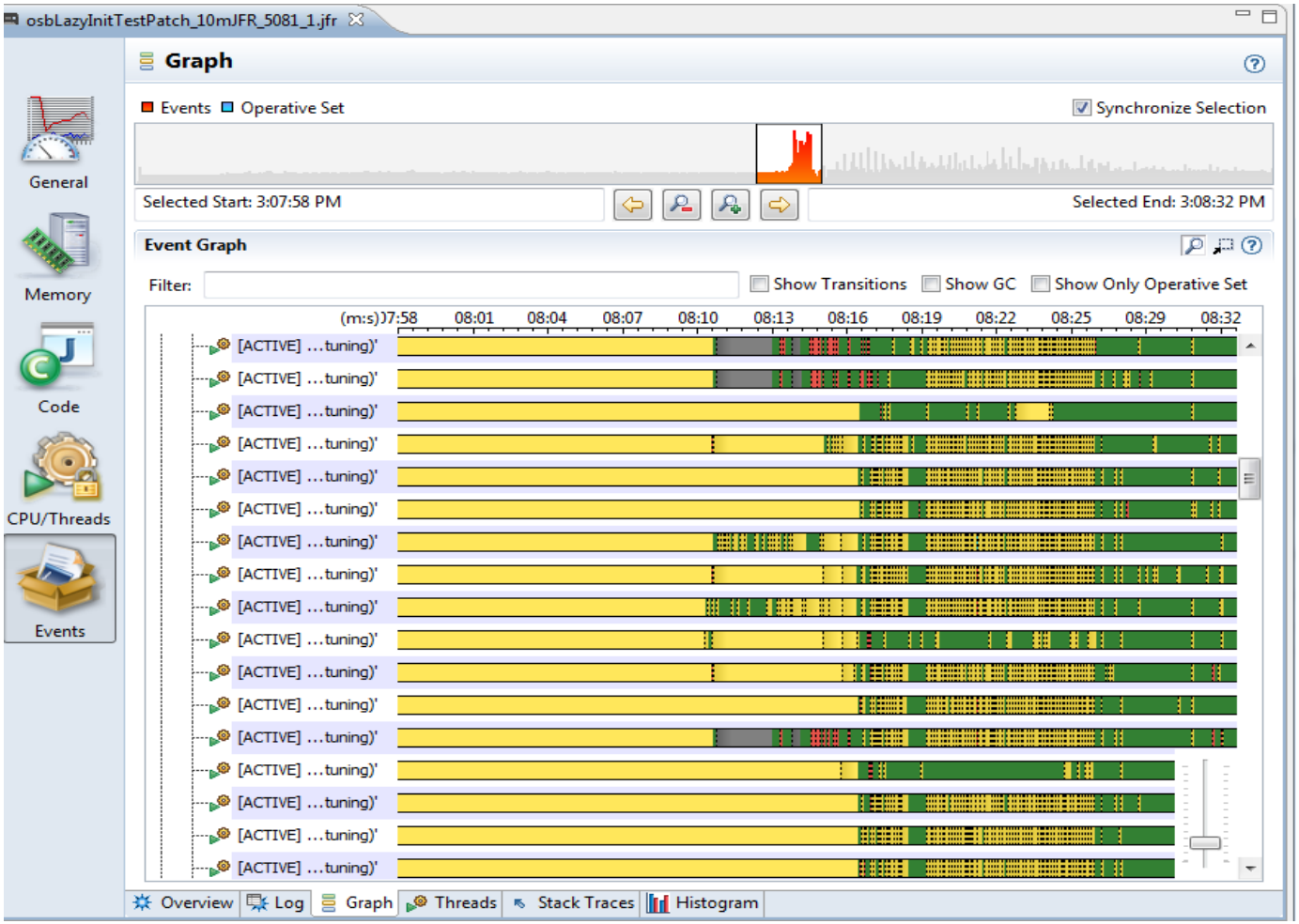


JVM 性能调优

JRockit Mission Control — 控制台

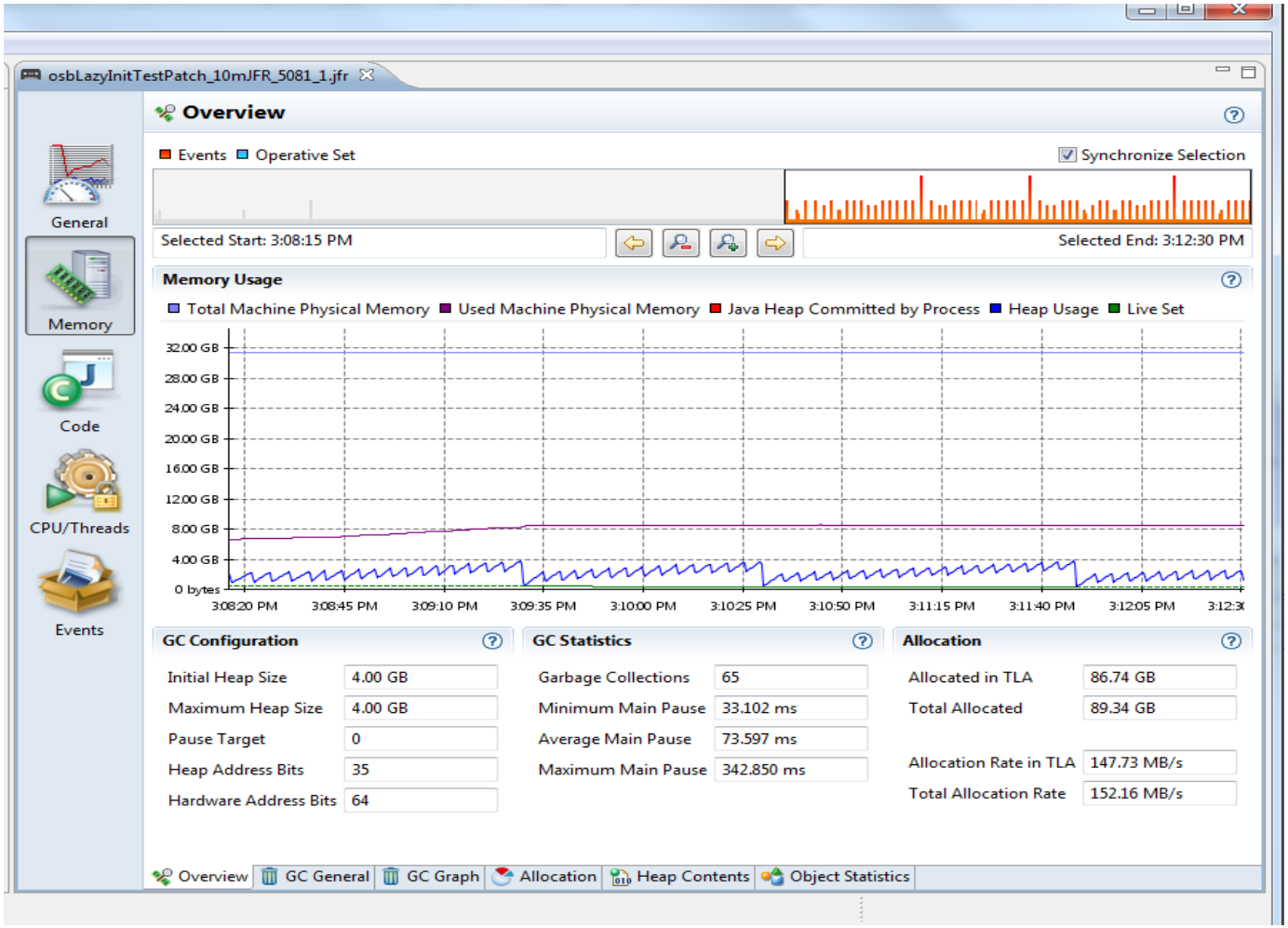
JVM 性能调优

JRockit Mission Control — JFR



JVM 性能调优

JRockit Mission Control — JFR



JVM 性能调优

GC 调优 — 常规

- 我们将探讨如何通过设置最少的参数来实现最高的性能
- 常规调优建议
  - 保持简单性
  - 提供基本参数（-X 参数）
    - -Xms、-Xmx、-Xmn
  - 选择一个 GC/性能优先级
    - 权衡吞吐量与暂停时间
  - 其余参数大多使用默认值
    - 让*人体工程机制*计算正确值
    - 仅当默认值无效时调优

JVM 性能调优

GC 调优参数 — 常规

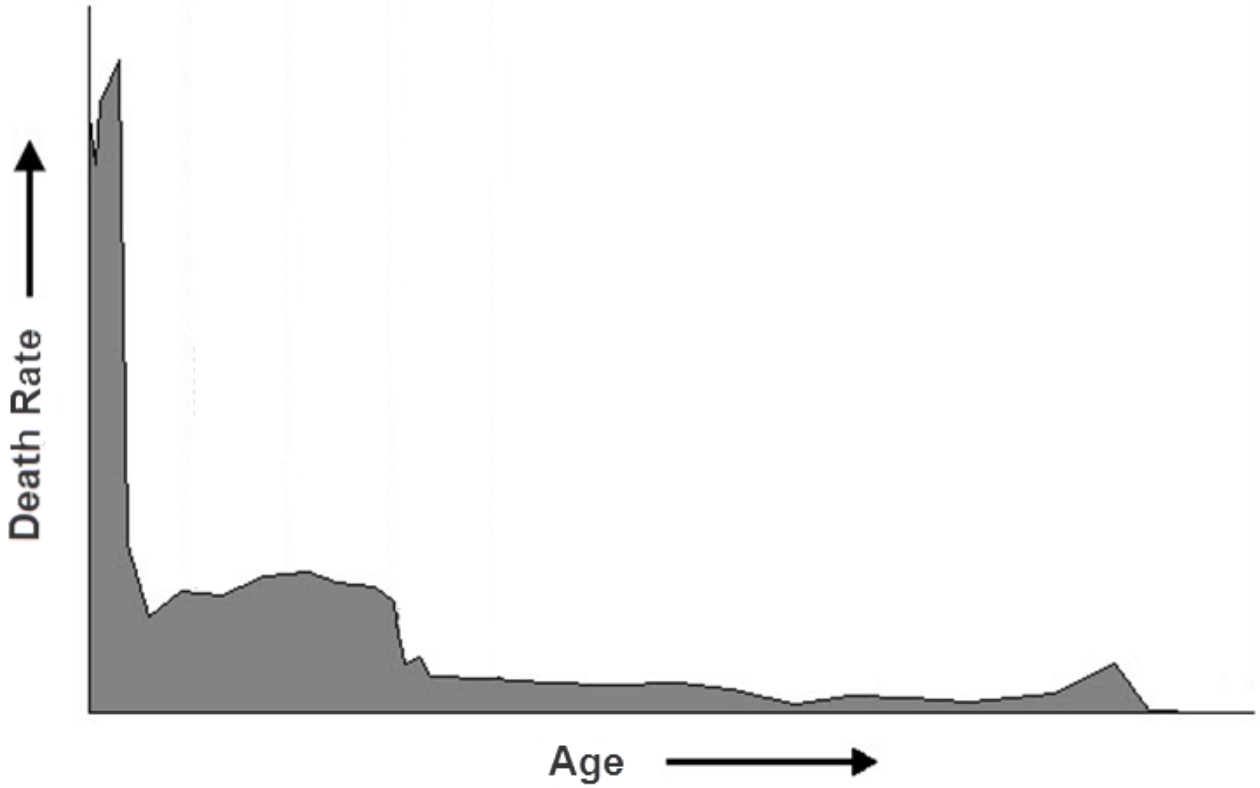
- 最常使用的 GC 参数
  - 垃圾收集策略（例如，不分代、并发、并行 GC 和确定性等）
  - GC 并行机制（串行、并行或并发）
  - 代大小
- 最常使用的 GC 诊断参数
  - GC 日志记录（例如 *-verbose:gc*）
  - 日志详细程度。例如 *-XX:+PrintGCDetails (*Hotspot*),* *–Xverbose:gcpause* (JRockit)
  - GC 日志文件。例如 *-Xloggc:<file> (*Hotspot), *-Xverboselog:(*JRockit)
  - ***在生产环境中启用 GC 日志记录是非常安全的***

议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

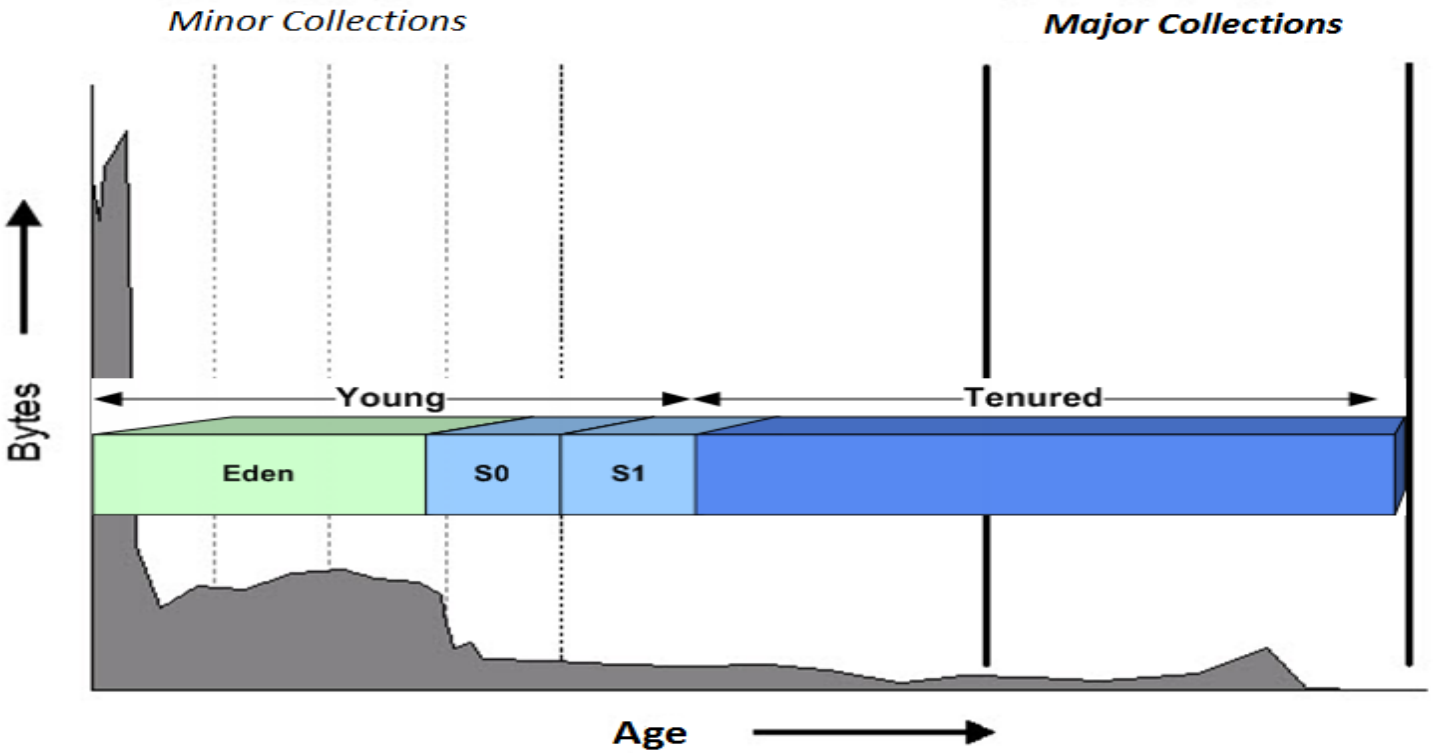
GC 基础知识

垃圾分布 — 对象夭亡



GC 基础知识

垃圾分布 — 分代回收



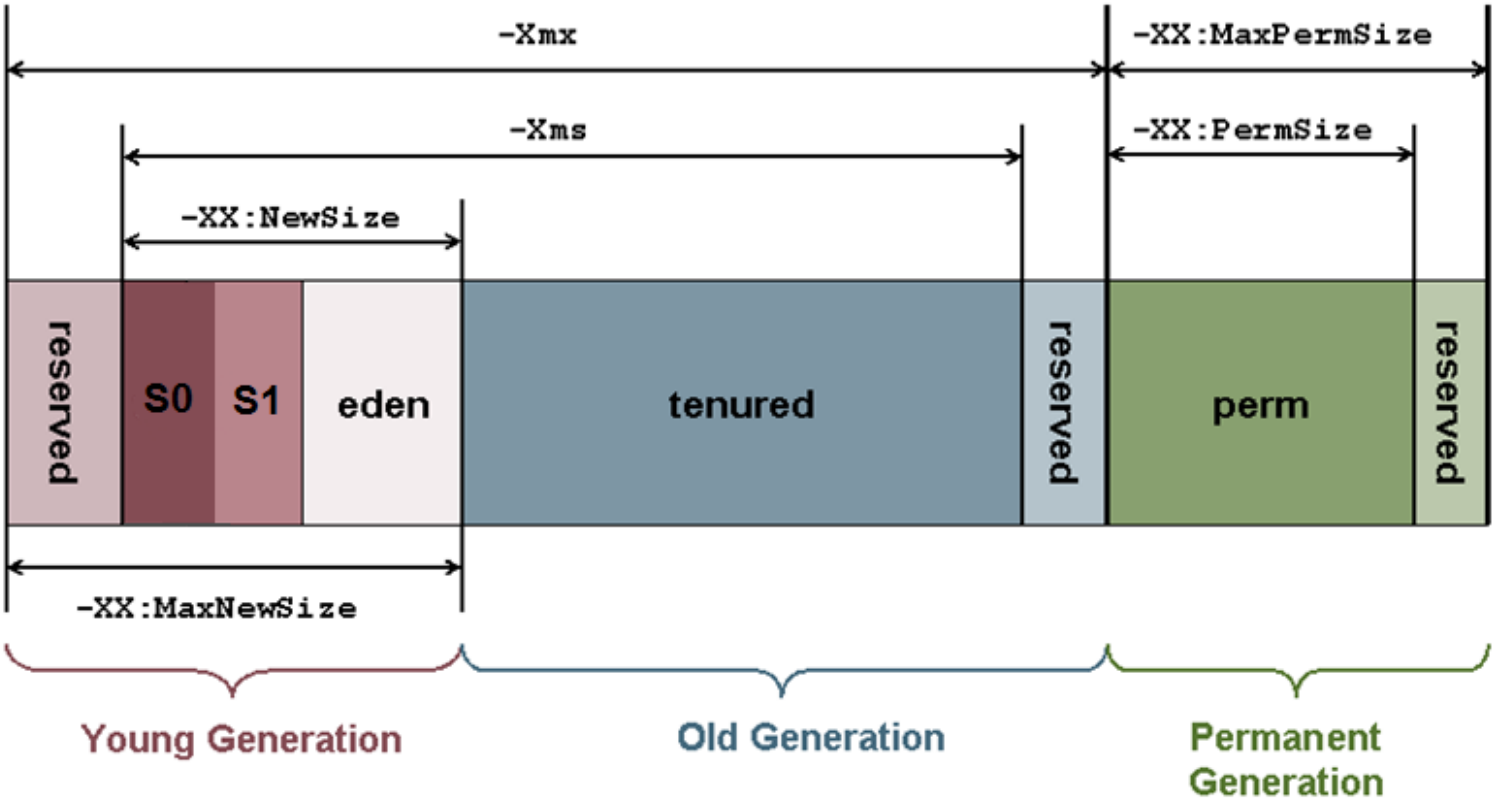
- 区段与分布曲线十分相符
- 使用最高效的算法回收各区段
- 调整区段大小，确保实现最佳 GC 性能

议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

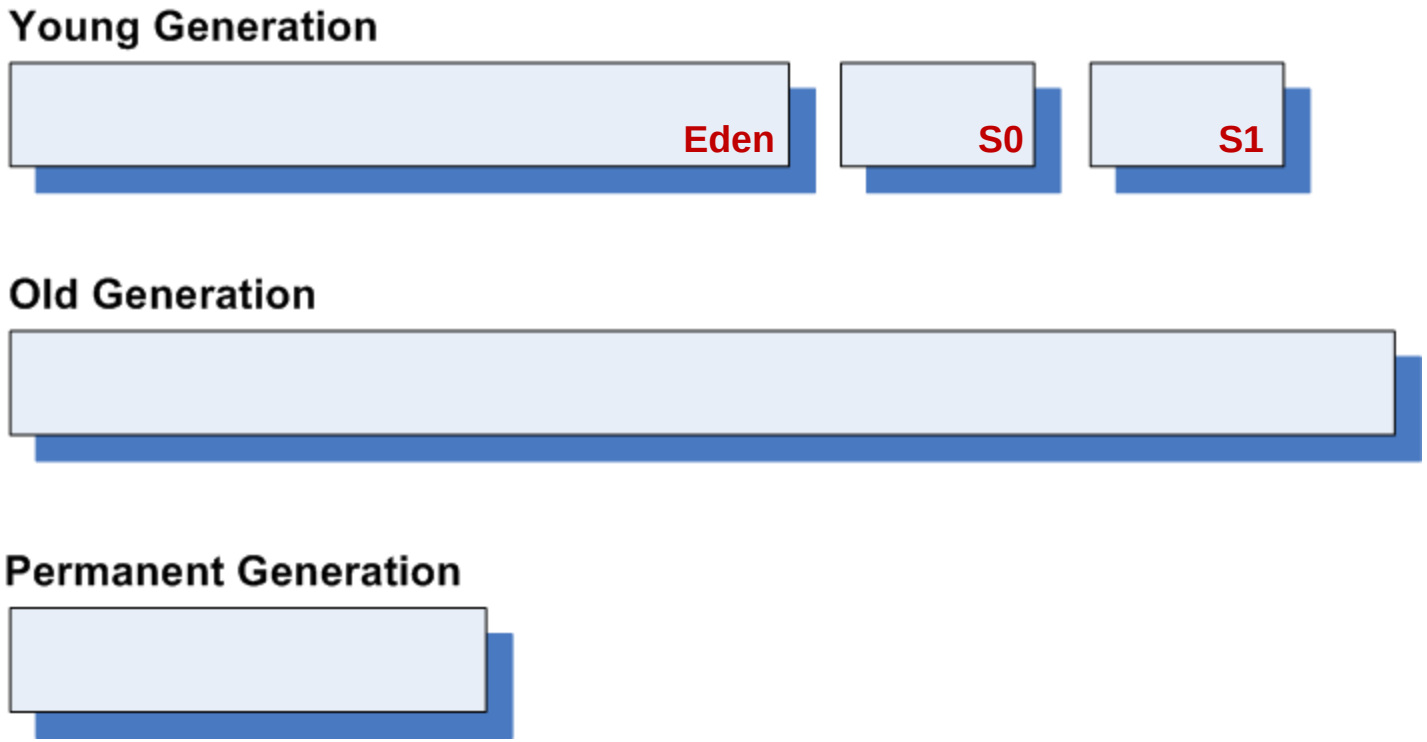
Hotspot 内部机制

Hotspot 堆



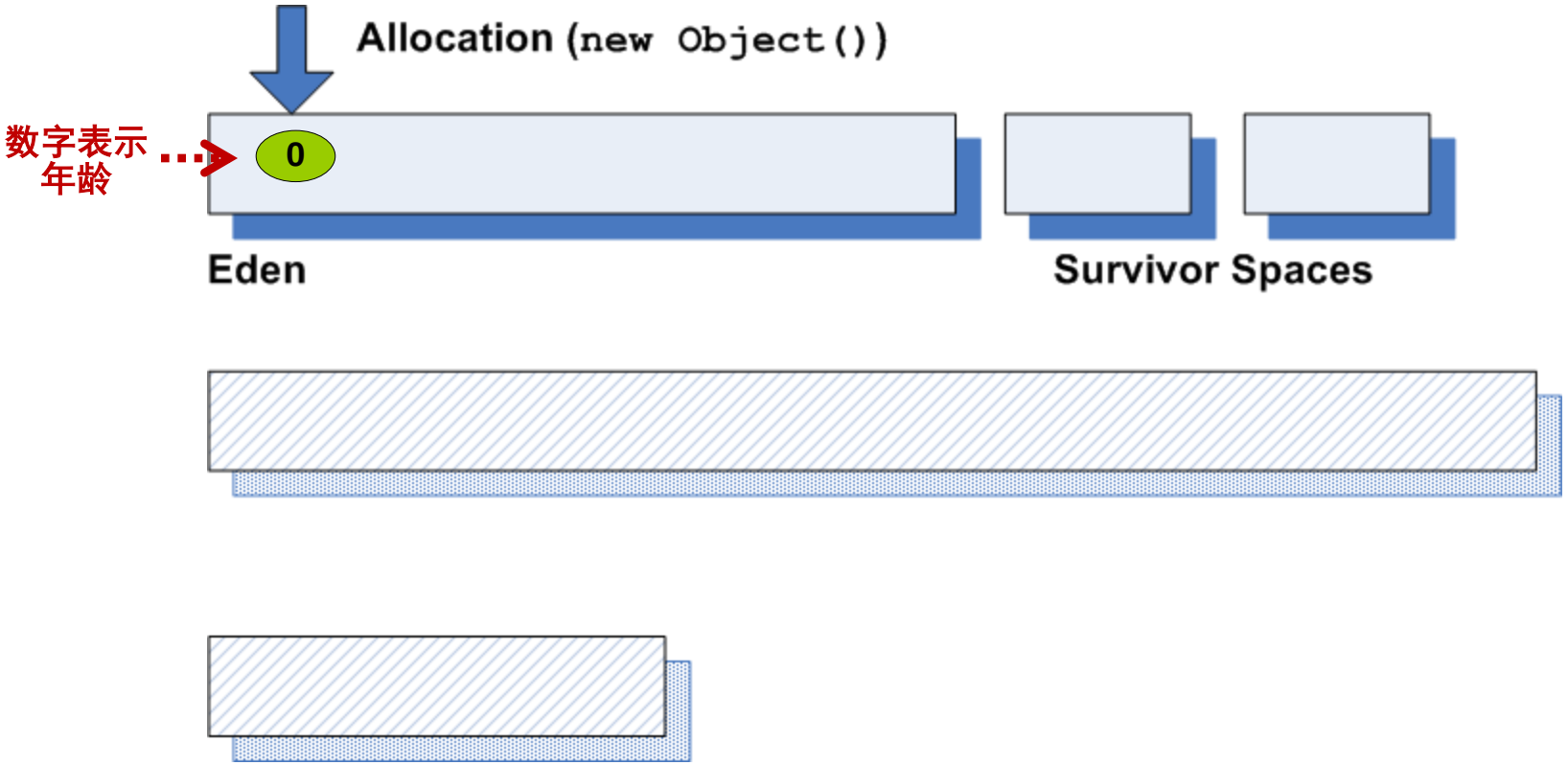
Hotspot 内部机制

代和对象生命周期



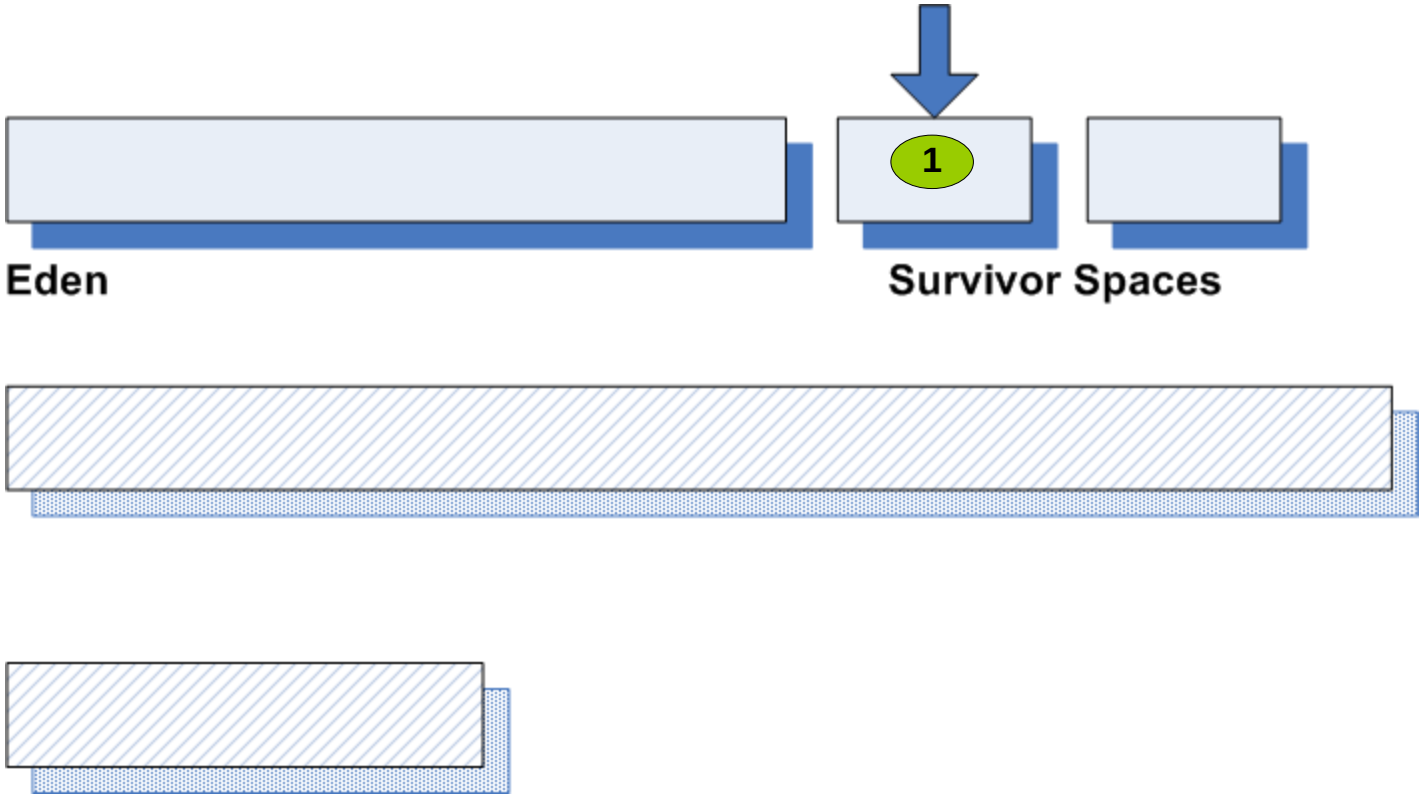
Hotspot 内部机制

对象生命周期



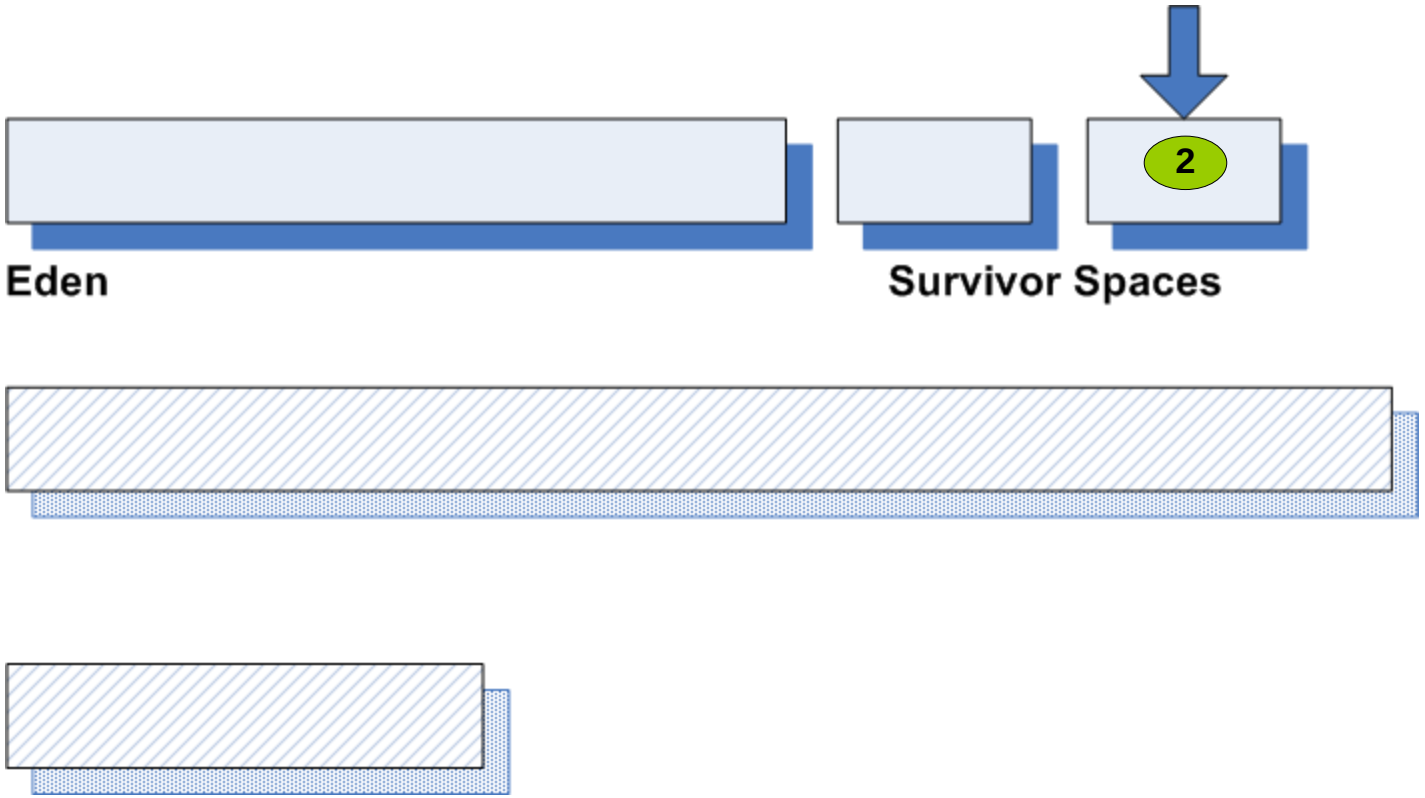
Hotspot 内部机制

对象生命周期



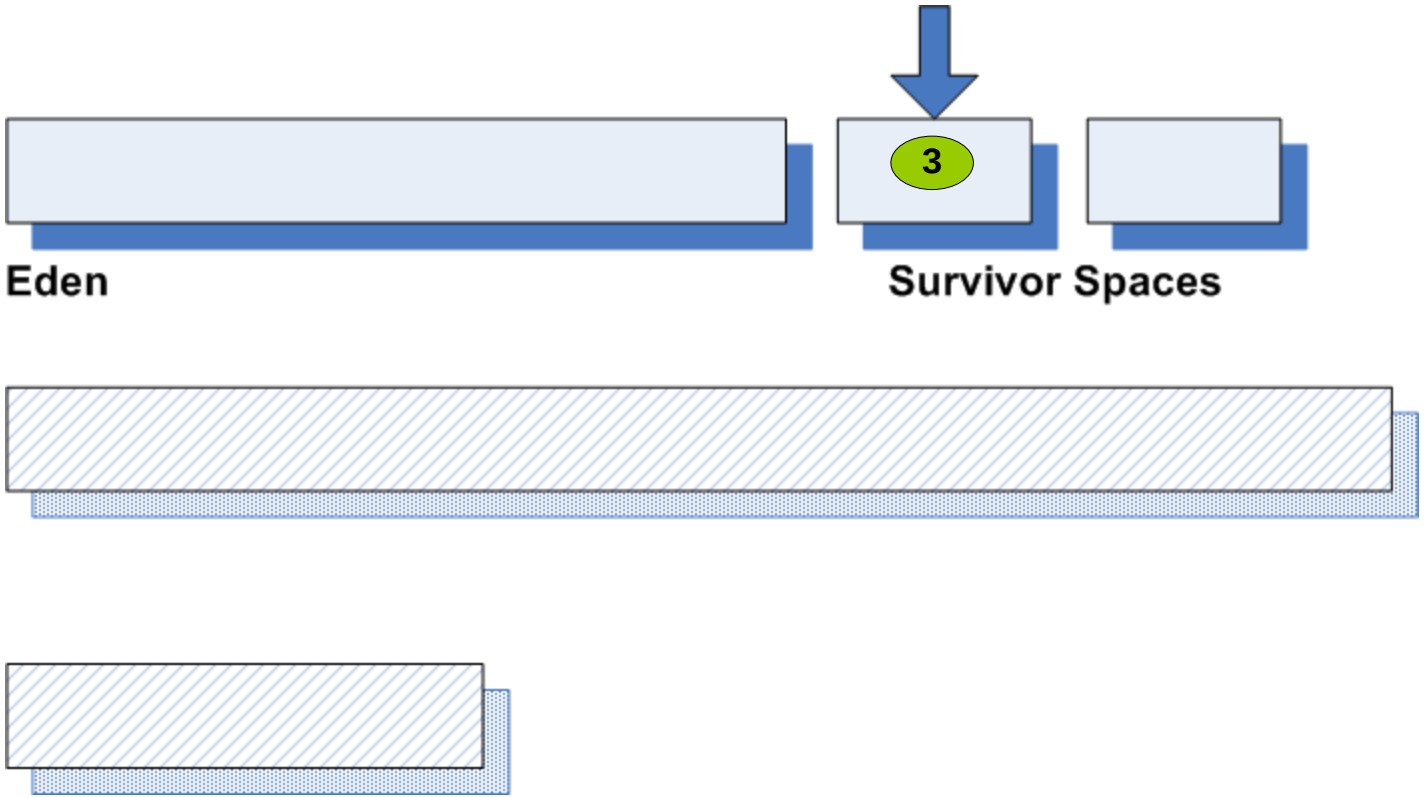
Hotspot 内部机制

对象生命周期



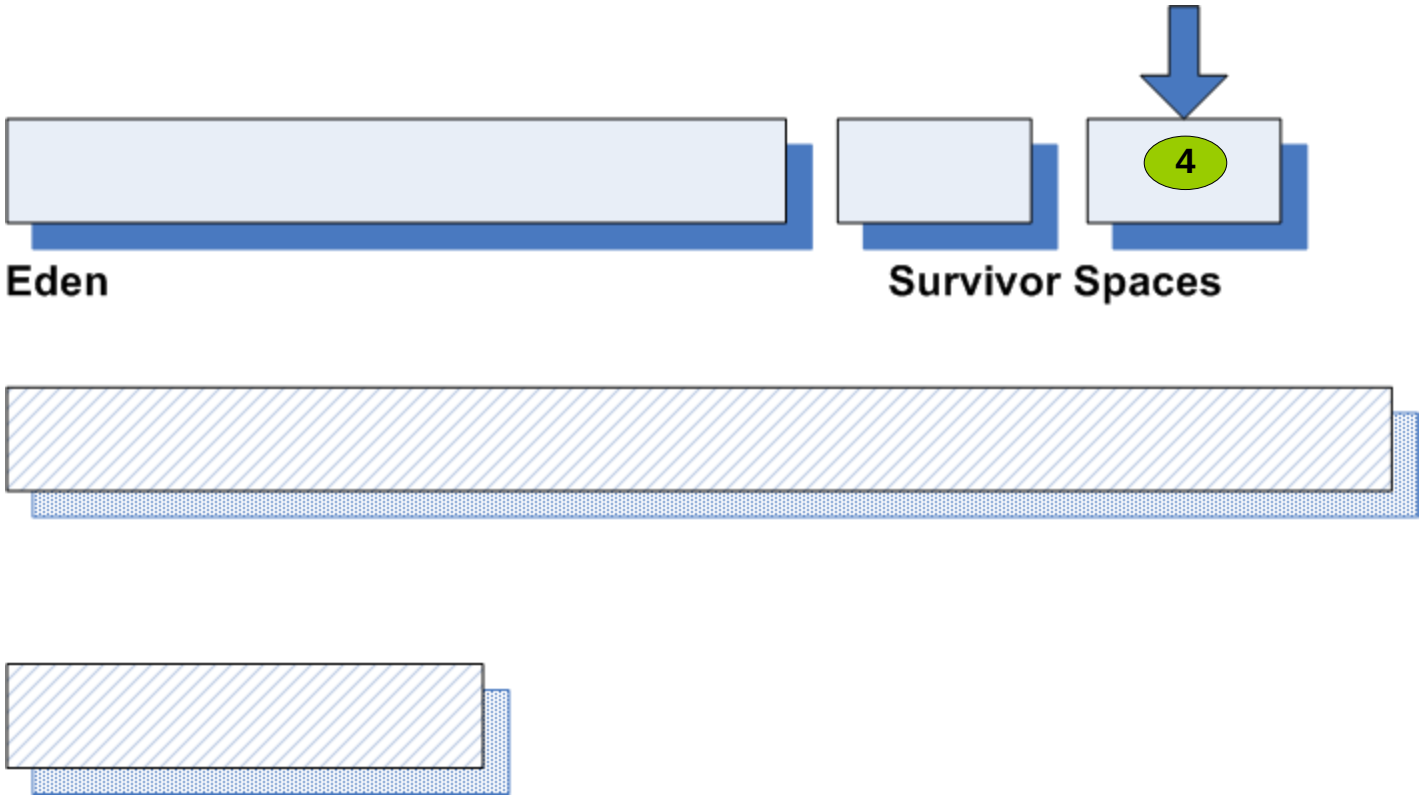
Hotspot 内部机制

对象生命周期



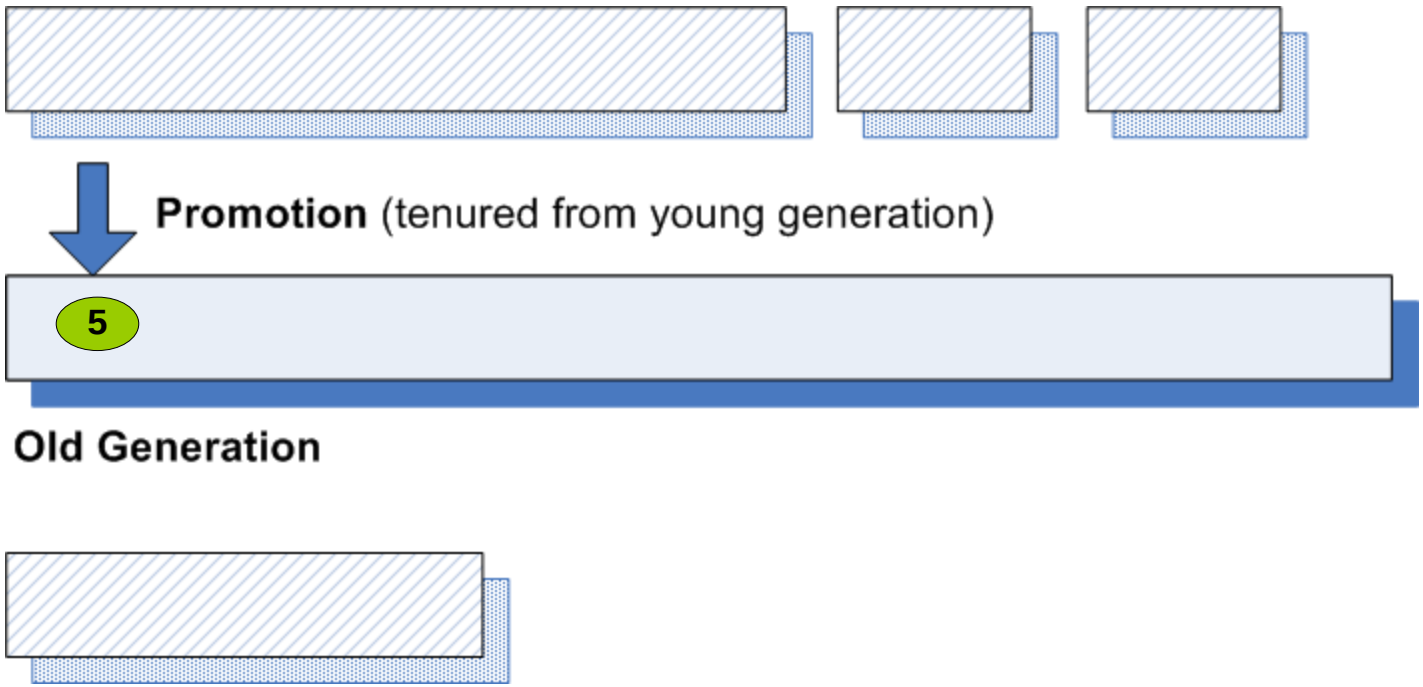
Hotspot 内部机制

对象生命周期



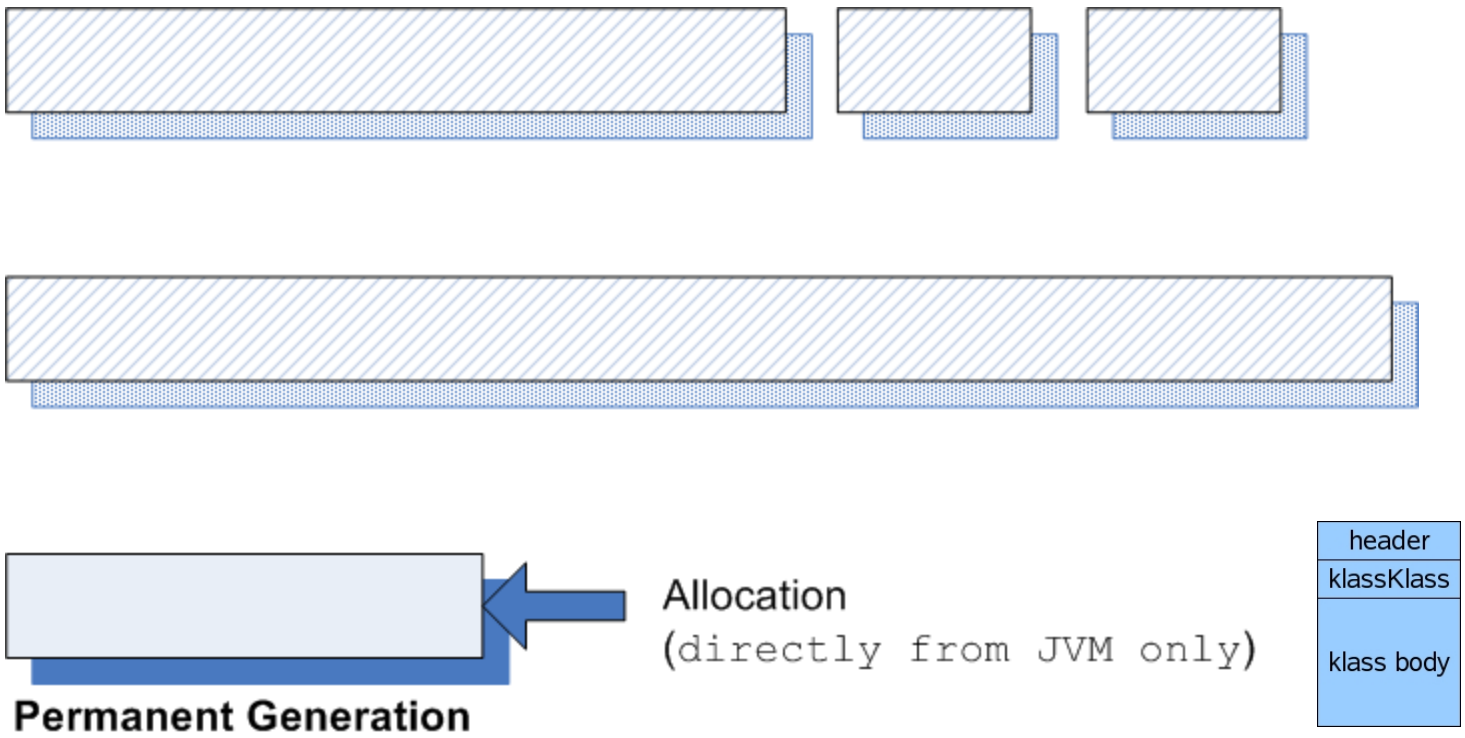
Hotspot 内部机制

对象生命周期



Hotspot 内部机制

对象生命周期

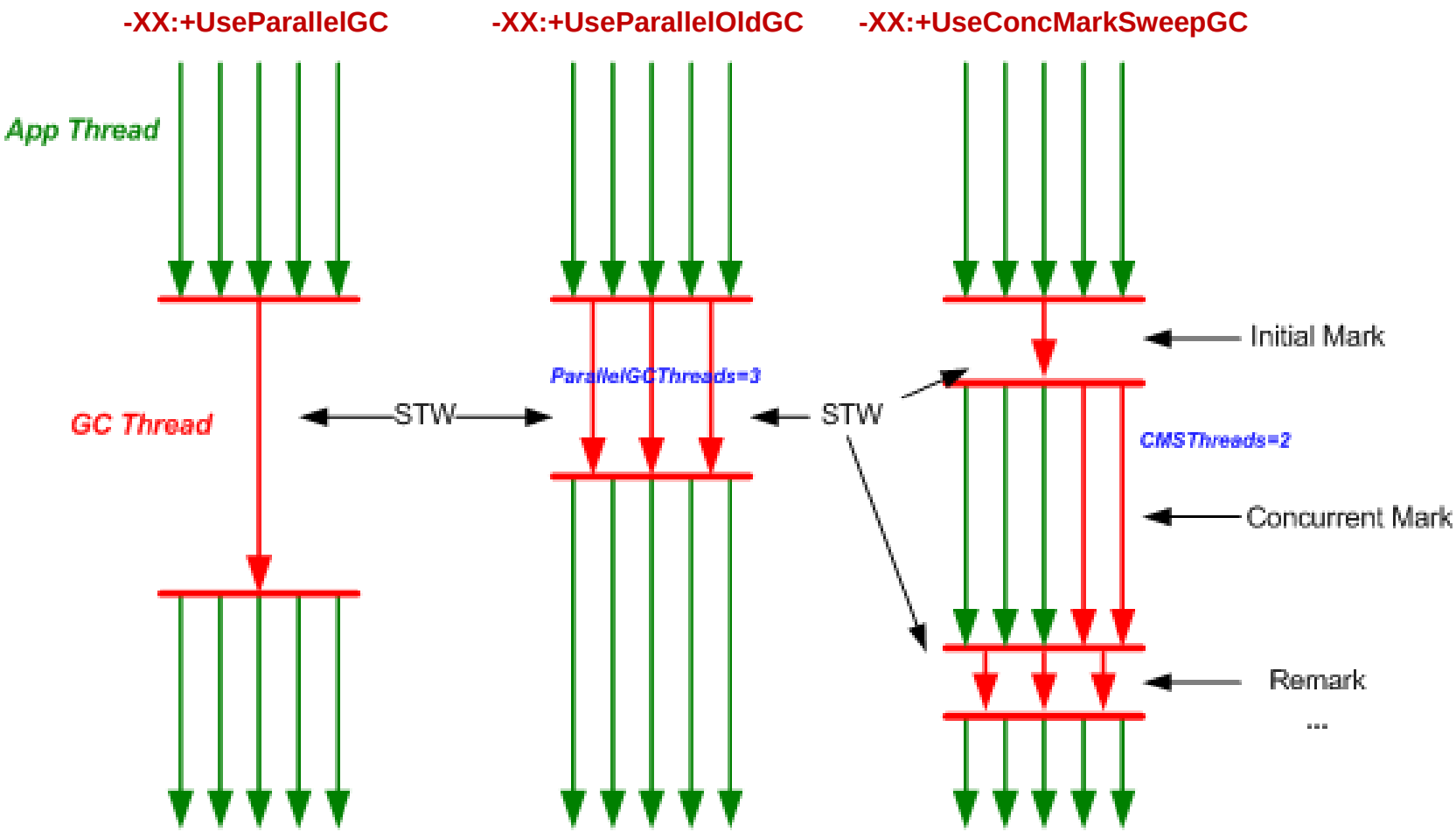


议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

Hotspot 调优

垃圾收集器 — 年老代收集



Hotspot 调优

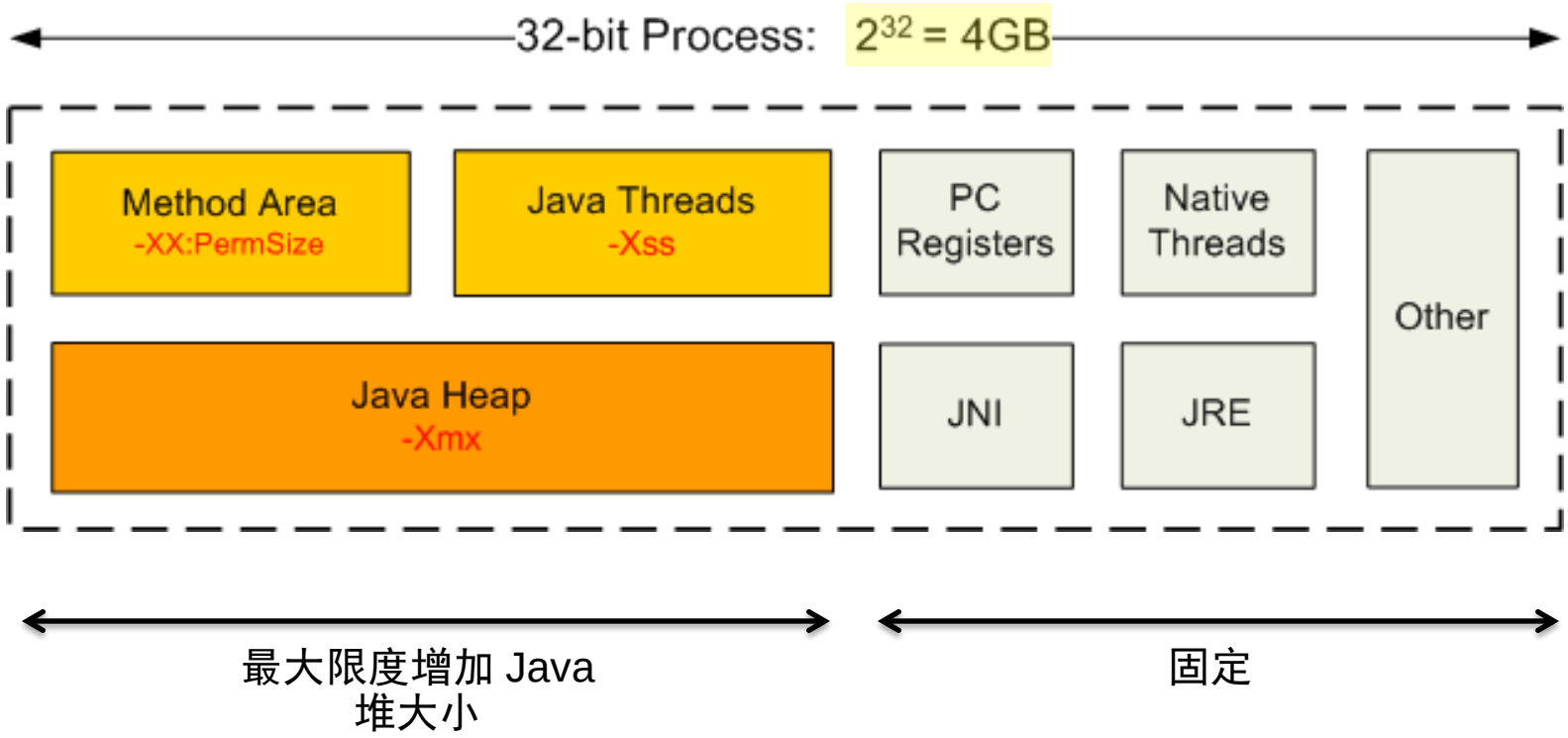
垃圾收集器

Hotspot 分代收集器

- 年轻代收集也称作“次要”收集
  - 使用副本收集（最高效的类型）
- 年老代收集也称作“主要”收集
  - 需要通过“完整 GC”来缩并零散的堆空间
  - 年老代收集包括
    - 串行年老代收集 (-XX:+UseParallelGC)
    - 并行年老代收集 (-XX:+UseParallelOldGC)
    - 并发年老代收集 (-XX:+UseConcMarkSweepGC)

Hotspot 调优

Hotspot 本机进程空间



Hotspot 调优

32 位与 64 位对比

- 32 位
  - 需要最大 2.5G/3G 左右的堆大小
  - 通过减小栈来最大限度增加堆大小 (-xss:128k)
- 64 位，带有/不带压缩引用
  - -XX:+/-**UseCompressedOops**（默认为 JDK6_18+）
  - 压缩引用：最大 32GB（最佳 26GB）
  - -Xmx：26G（压缩后）/不受限（常规）
- 32 位 → 64 位迁移
  - 需要更大的堆大小（20% 左右）
  - 吞吐量影响微小（无压缩引用）
- 64 位是当今服务器的优先选择
  - 自 Fusion Middleware 12c 发布之后的唯一选择

Hotspot 调优

调整堆大小

- 年轻代的大小将决定
  - 次要 GC 的***频率***
  - 次要 GC 收回的***对象数量***
- 年老代大小
  - 应达到应用程序***稳定状态***的 实时数据大小
  - 尝试***最大限度减小***主要 GC 的频率
- JVM 内存占用不应超过物理内存
  - 最大达到 RAM 的 80-90%（为操作系统留出空间）
- **经验法则：**应尽量增加年轻代收回的对象。尽量增加完整 GC 频繁

Hotspot 调优

调整堆大小（续）

- 重新调整任何代的大小都需要 Full GC
- Set –Xmx = –Xms
  - 防止堆大小 (Full GC) 从 Xms 增大到 Xmx
  - 性能更优
  - 并非总是生产可用性的最佳选择（OOME 更合适使用内存交换）
- 持久代大小
  - *-XX:PermSize = -XX:MaxPermSize*
  - 持久代占用空间大小难以预测
  - 设置足够高以防止 PermGen OOME
- 将 -XX:NewSize 设置为 -XX:MaxNewSize
  - 优先使用 –Xmn

Hotspot 调优
并行 GC 线程

- 并行 GC 线程的数量由以下参数控制
  - *-XX:ParallelGCThreads=<num>*
  - 默认值假定只有 1 个 JVM
- 调整以下要素的 *ParallelGCThreads* 值
  - 部署在系统/虚拟机上的 JVM 数量
  - CPU 芯片架构和内核，例如 Sun CMT、Intel Hyperthreads
- 示例：
  - Exalogic 计算节点搭载 2 个支持超线程技术（每个内核两个线程）的 6 核 Intel CPU，相当于 24 个虚拟 CPU。
  - 如果每个节点运行 4 个 WLS 实例
    - 24/4 = 6
    - 将 *–XX:ParallelGCThreads* 设置为小于或等于 6，作为各 WLS JVM 的起点

Hotspot 调优

CMS 收集器调优

- 并发 Mark Sweep（短停顿）收集器
  - *-XX:+UseConcMarkSweepGC*
- 优点：
  - 最差情况下的延迟优于吞吐量收集器
- 缺点：
  - 应用程序吞吐量低于吞吐量收集器
  - 分散化 — GC 周期更长（尽管支持并发）
- 年老代大小至少增加 20% 到 30%
- 根据所述信息调优年轻代
- 需要更谨慎地避免过早提升
  - CMS 中的提升成本非常高
  - 导致碎片增加
  - ***Full GC 不可避免***

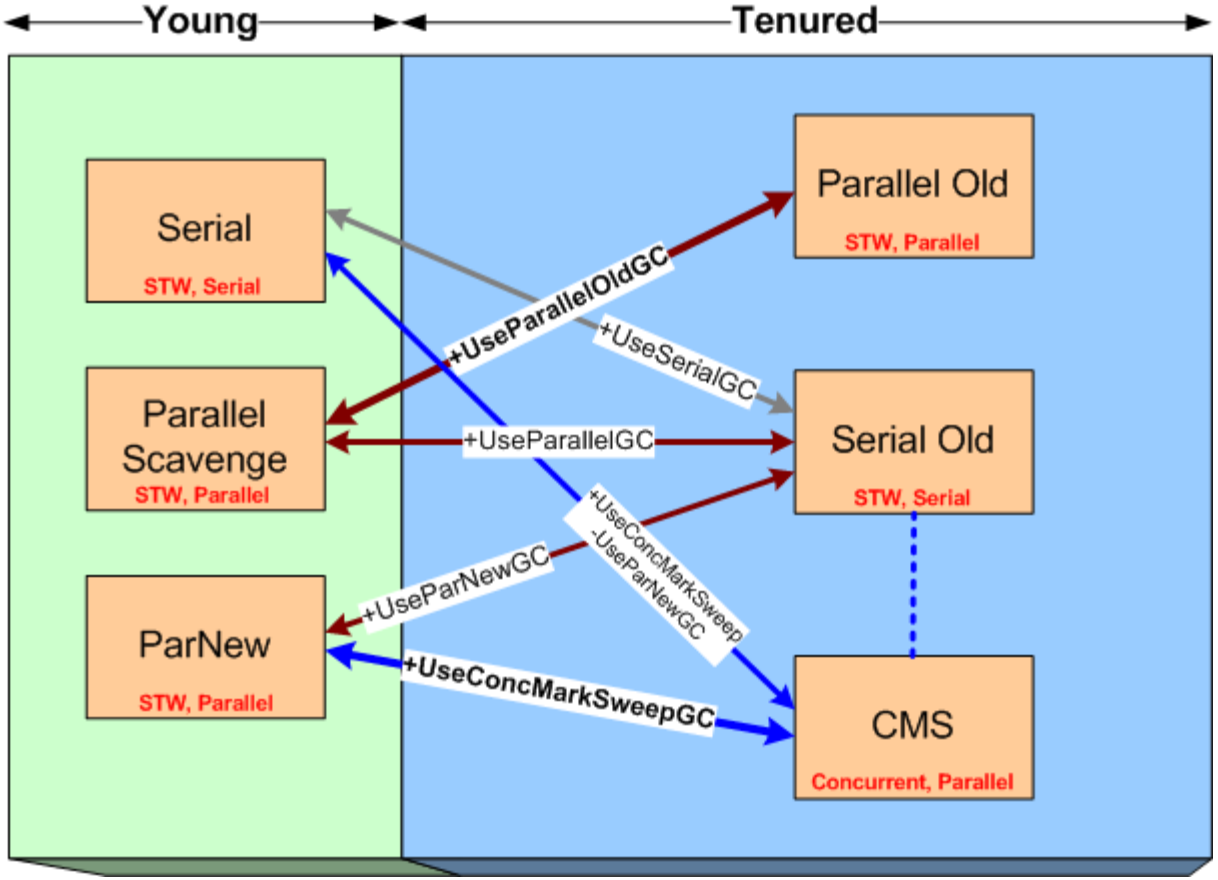
Hotspot 调优

CMS 启动阈值

- 过早开始 CMS 周期
  - 频繁的 CMS 周期
- 过晚开始 CMS 周期
  - 可能会发生清空失败/Full GC
  - 过早比过晚更加安全
- 默认的 CMS 启动阈值
  - 动态计算
  - 几乎总是开始过晚（CMS 错失）
- 覆盖默认值
  - **-XX:CMSInitiatingOccupancyFraction=<percent> （例如 50）**
    - 设置足够低以防止“*并发模式故障*”
    - 设置 ***-XX:+UseCMSInitiatingOccupancyOnly***，确保始终使用

Hotspot 调优

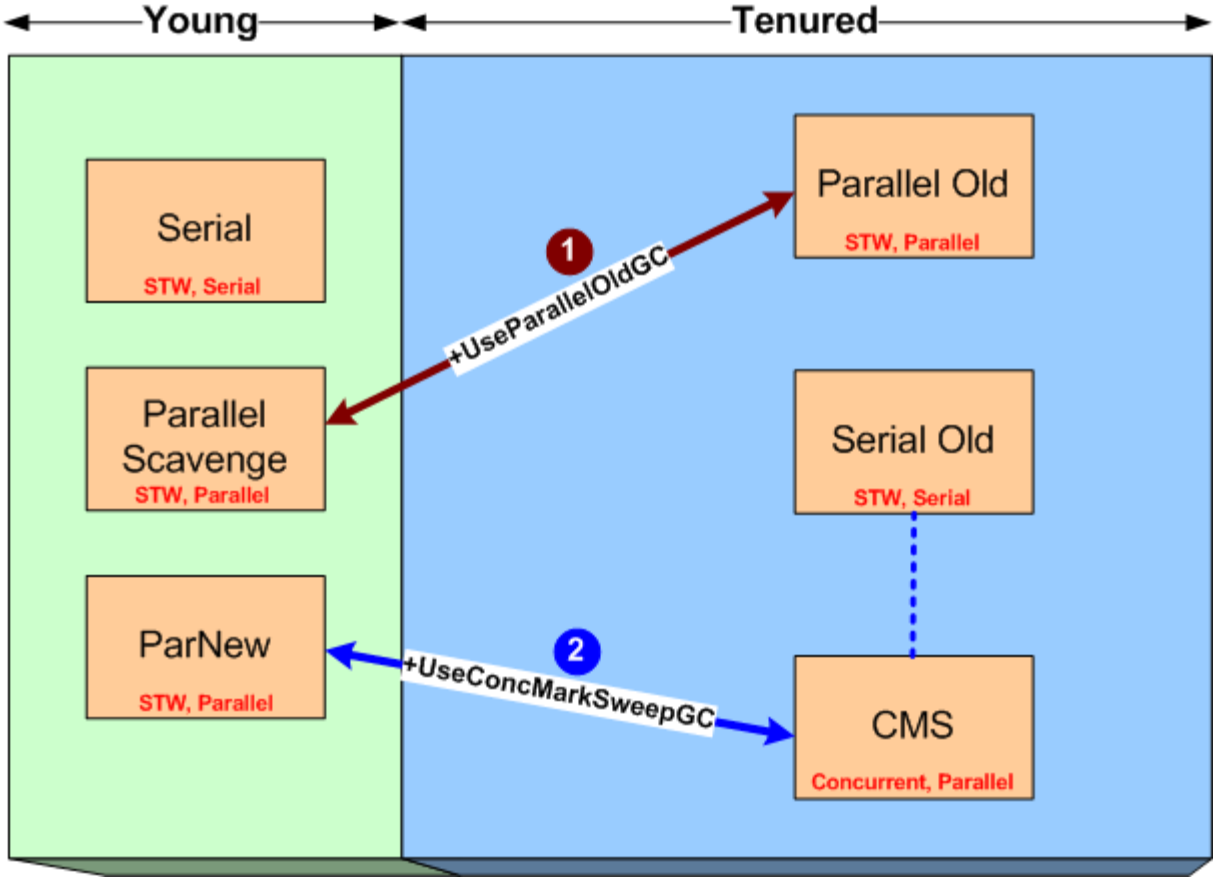
有效的 GC 组合



信息来源：Jon Masamitsu 的博客

Hotspot 调优

GC 建议



信息来源：Jon Masamitsu 的博客

Hotspot 调优

默认 GC 值 — JDK6

默认值	建议
-XX:+UseParallelGC	-XX:+UseParallel**Old**GC
ParallelGCThreads=CPU	ParallelGCThreads=CPU 数量**/JVM 数量**
SurvivorRatio=32	SurvivorRatio=**8**
PermSize=64M	PermSize=**MaxPermSize**
无 GC 日志记录	**Verbose** GC 日志记录

Hotspot 调优

大页面

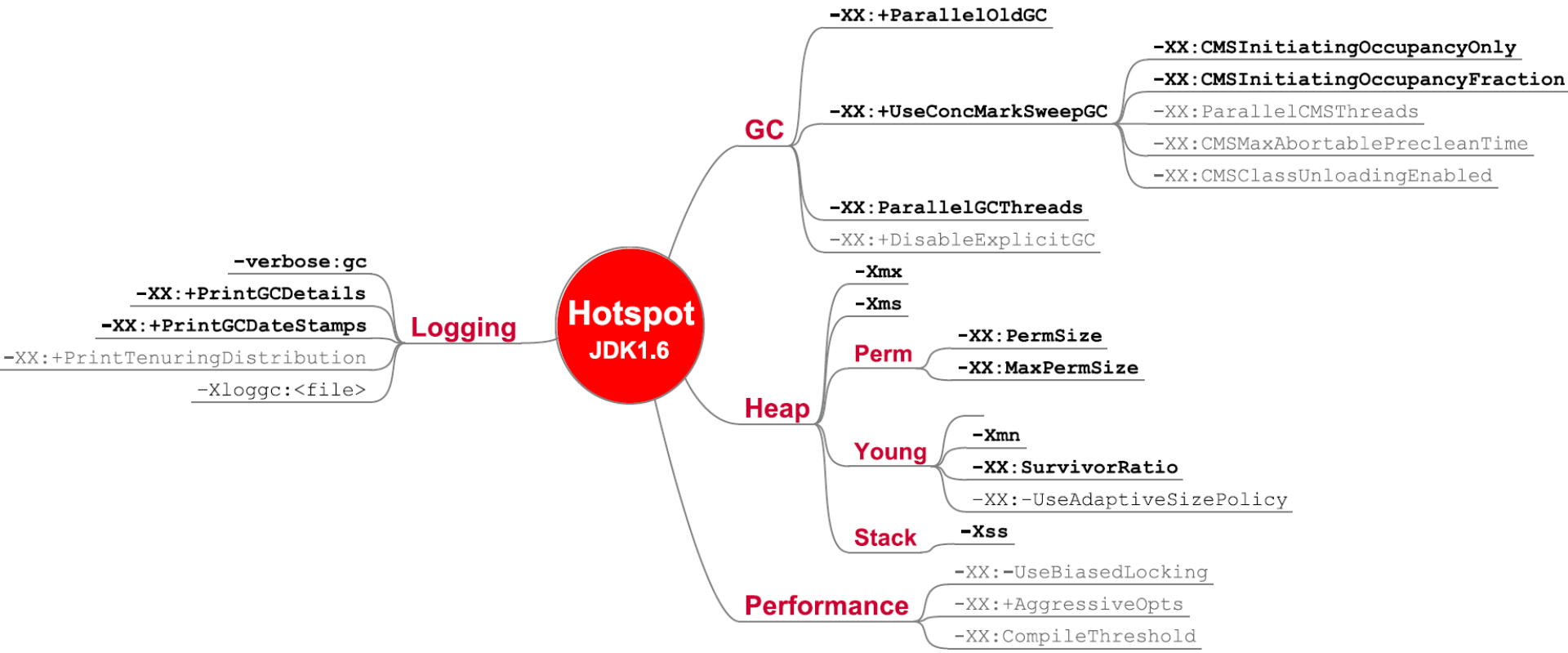
- 使用大页面作为 OS VM 页面
  - 内存密集型应用程序
  - 使用大对象的应用程序
- 在 Hotspot 上使用大页面
  - **-XX:+UseLargePages**
    - 在 Solaris 上，默认为“启用”
  - **-XX:LargePageSizeInBytes=4m**
    - 有效值因 OS 和芯片架构而异
    - 例如，x86 上通常最大可支持 2m

Hotspot 调优
大页面示例

- Exalogic 和 Linux 大页面
  - Intel 芯片支持大页面（最大 2m）
  - 常规页面大小 = 2KB
  - Exalogic 上的现成大页面配置
    - 大页面大小 = 2MB
    - 大页面数量 = 10,000
    - 总共为大页面预留 20GB 空间
      - 常规页面的预留空间为 (96GB – 20GB)
    - 修改“*vm.nr_hugepages*”实现重新平衡
- 在 JRockit 上使用 ***-XX:+UseLargePagesForHeap***
    - 20GB 大页面块上现已分配 JVM 堆
- 性能提升多达 15%

Hotspot 调优

Hotspot 调优快速参考



议题

- JVM 基础知识
- JVM 性能调优
- GC 基础知识
- Hotspot 内部机制
- Hotspot 调优
- 诊断 GC 问题

诊断 GC 问题

读取 GC 日志 — 次要 GC

- **GC 诊断*需要*GC 日志**

```
[GC[PSYoungGen: 99952K->14688K(109312K)]
  422212K->341136K(764672K), 0.0631991 secs]
  [Times: user=0.06 sys=0.00, real=0.06 secs]
```

- “PSYoungGen”表示这是一次次要收集
  - 其中使用了吞吐量收集器（例如 UseParallelGC）
- 数字表示 *SIZE_BEFORE_GC->SIZE_AFTER_GC(MAX_SIZE)*
  - STW 暂停时间 = 0.06 秒（“实”时）
- 类似的 CMS 次要收集日志会显示“ParNew”

```
[GC[ParNew: 99952K->14688K(109312K)]
  422212K->341136K(764672K), 0.0631991 secs]
  [Times: user=0.06 sys=0.00, real=0.06 secs]
```

诊断 GC 问题

读取 GC 日志 — Full GC

- 吞吐量收集器“Full GC”

```
[Full GC[PSYoungGen: 11456K->0K(110400K)]
   [ParOldGen: 651536K->58466K(655360K)]
    662992K->58466K(765760K)
   [PSPermGen: 10191K->10191K(22528K)], 1.1178951
   secs[Times: user=1.01 sys=0.00, real=1.12 secs]
```

- CMS 收集器“Full GC”

```
[Full GC 59.550: [CMS59.608: [CMS-concurrent-sweep:
0.189/0.191 secs] [Times: user=0.37 sys=0.00, real=0.19
secs]
(concurrent mode failure): 1048575K->1048575K(1048576K),
3.4256231 secs] 2936061K->2206984K(2936064K), [CMS Perm
: 2621K->2621K(524288K)], 3.4263668 secs]
[Times: user=3.35 sys=0.00, real=3.42 secs]
```

诊断 GC 问题

GC 日志分析

- **Full GC 的暂停时间通常很长**
- **在详细 GC 日志中搜索“Full GC”**

日志中的 Full GC	操作
采用堆转储？	忽略此 Full GC（由堆转储触发）
显示字符串“Full GC (system)”	设置 *-XX:+DisableExplicitGC*
显示持久代已满	增加 *–XX:MaxPermSize*
显示大小调整后的堆	设置 *–Xmx=-Xms*、*-XX:NewSIze=-XX:MaxNewSIze*
显示“concurrent mode failure”	*减小 -XX:CMSInitiatingOccupancyFraction* *或者* *不再使用 –XX:+CMSInitiatingOccupancyOnly*

Full GC

```
 20416.613: [CMS-concurrent-sweep-start]
20420.628: [CMS-concurrent-sweep: 4.004/4.015 secs]
20420.628: [CMS-concurrent-reset-start]
20420.892: [CMS-concurrent-reset: 0.264/0.264 secs]
20422.176: [Full GC 20422.177: [CMS (concurrent mode failure):
1815018K->912719K(1835008K), 18.2639275 secs] 1442583K-
>912719K(2523136K), [CMS Perm : 202143K->142668K(262144K)], 18.2649505
secs]
```

- 堆在 CMS 完成之前耗尽
- 动态调整的 CMS 启动占用空间大小不正确
- 手动指定一个更加保守的启动占用空间大小
  - -XX:+UseCMSInitiatingOccupancyOnly
  - -XX:CMSInitiatingPermOccupancyFraction=<percent>

Full GC

```
429417.135: [GC 429417.135: [ParNew: 1500203K->100069K(1747648K),
0.3973722 secs] 3335352K->1935669K(3844800K), 0.3980262 secs] [Times:
user=0.85 sys=0.00, real=0.40 secs]
430832.180: [GC 430832.181: [ParNew: 1498213K->103052K(1747648K),
0.3895718 secs] 3333813K->1939101K(3844800K), 0.3902314 secs] [Times:
user=0.83 sys=0.01, real=0.39 secs]
431370.238: [Full GC 431370.238: [CMS: 1836048K->1808511K(2097152K),
43.4328330 secs] 2481043K->1808511K(3844800K), [CMS Perm : 524287K-
>475625K(524288K)], 43.4336938 secs] [Times: user=40.13 sys=0.73,
real=43.43 secs]
```

- Full GC 由 PermGen 耗尽触发
  - 年老代未接近满额，但触发 Full GC
  - Perm 在 Full GC 之前已满
- 解决方法：
  - 增加 *–XX:MaxPermSize*
  - 使用 *–XX:+CMSClassUnloadingEnabled*

Full GC

```
39195.195: [Full GC (System) 39195.195: [CMS: 641844K-
>617525K(1318912K), 27.0243921 secs] 751876K->617525K(1698624K), [CMS
Perm : 205856K->205495K(421888K)], 27.0250058 secs] [Times: user=69.89
sys=0.05, real=27.03 secs]
39222.431: [Full GC (System) 39222.431: [CMS: 617525K-
>612104K(1318912K), 25.8235298 secs] 639071K->612104K(1698624K), [CMS
Perm : 205498K->205495K(421888K)], 25.8240855 secs] [Times: user=51.70
sys=0.02, real=25.82 secs]
```

- 连续的 Full GC
- Full GC 日志中有“(System)”
- 解决办法：
  - 让客户删除代码中的 `system.gc()`
  - 添加 *“-XX:+DisableExplicitGC”*标记

Full GC

```
296.544: [GC [PSYoungGen: 736K->64K(832K)] 96847K->96191K(1023808K), 0.0013899
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.546: [GC [PSYoungGen: 703K->64K(832K)] 96831K->96207K(1023808K), 0.0007021
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.547: [GC [PSYoungGen: 101K->32K(832K)] 96244K->96199K(1023808K), 0.0005676
    secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
296.548: [Full GC [PSYoungGen: 32K->0K(832K)] [PSOldGen: 96167K-
    >56751K(1022976K)] 96199K->56751K(1023808K) [PSPermGen: 8115K->8115K(51200K)],
    0.0189618 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]
```

- 正常 Full GC 日志，无明确触发器（各代中都留有大量堆）
  - 用户本来可以触发堆转储
  - 堆转储导致在文件转储之前执行 Full GC

Full GC 暂停时间 — 外部因素

```
[Full GC 957910K->747933K(1004928K), 0.0077580 secs]
[Full GC 959079K->747525K(1004928K), 0.0069880 secs]
[Full GC 959014K->748193K(1004928K), 4.8153540 secs]
[Full GC 916083K->697827K(1004928K), 14.8503310 secs]
[Full GC 831689K->657890K(1008320K), 14.5647330 secs]
[Full GC 893862K->688939K(1004928K), 7.4950890 secs]
[Full GC 385884K->240312K(1004928K), 91.2939710 secs]
```

- 堆大小只有 1G
- 对于这种大小的堆来说，Full GC 耗时 91 秒过长
- 检查 OS 崩溃
  - 内存：过量分页
  - CPU： ~100%

问答