引言

本文通过理论加实践的方式逐步讲解JVM调优。共分四部分，第一部分：概述，主要讲解关于JVM调优的理论概念知识；第二部分：JVM调优工具，主要讲述在调优过程中所使用的工具以及命令；第三部分：JVM参数，主要讲述JVM主要调优参数；第四部分：案例分析，通过前三部分的理论基础结合真实项目案例来看看如何进行调优。

JVM调优基础

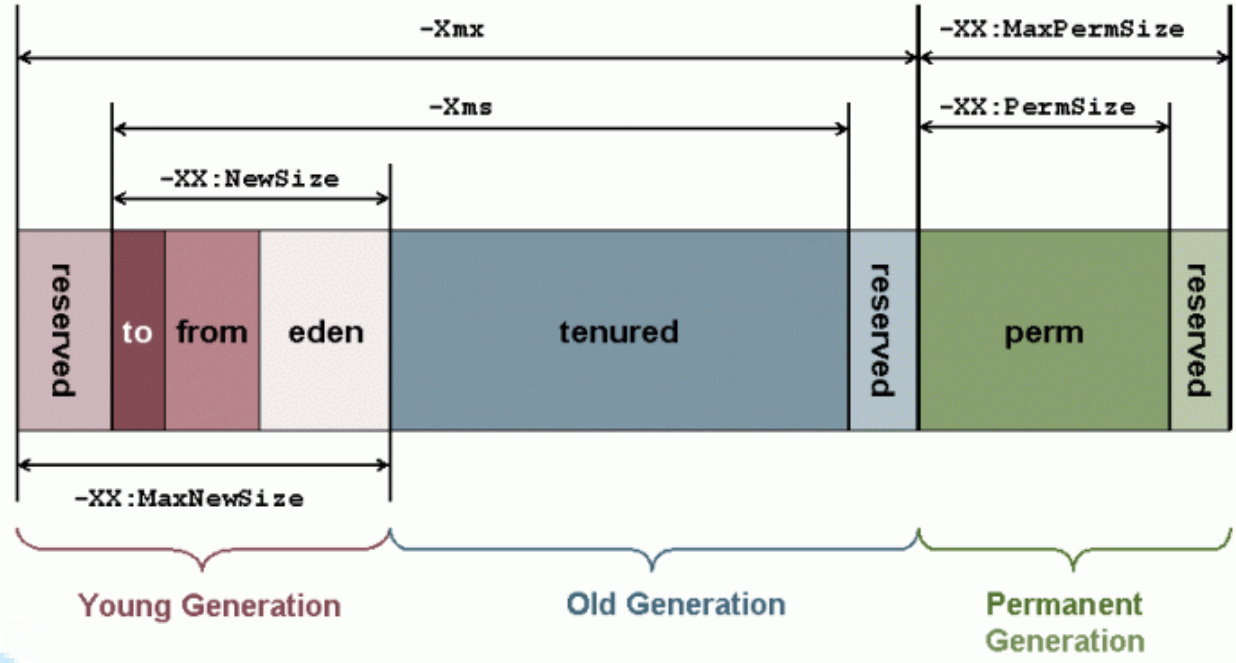
本章主要通过JVM基础、调优层次、调优指标、何时JVM调优、JVM调优目标、JVM调优原则以及JVM调优步骤介绍，以达到对JVM调优有个整体上的认识，后续通过具体的案例结合理论来看下如何进行调优。

1 JVM基础

在讲JVM调优之前，先简单回顾下JVM相关的基础知识，这里我们重点回顾下JAVA堆、垃圾回收器。这两块也是在JVM调优过程中重点关注的部分。

1.1 java堆

被所有线程共享，在虚拟机启动时创建，用来存放对象实例，几乎所有的对象实例都在这里分配内存。对于大多数应用来说，Java堆（Java Heap）是Java虚拟机所管理的内存中最大的一块。Java堆是垃圾收集器管理的主要区域，因此很多时候也被称做“GC堆”。如果从内存回收的角度看，由于现在收集器基本都是采用的分代收集算法，所以Java堆中还可以细分为：新生代和老年代；新生代又有Eden空间、From Survivor空间、To Survivor空间三部分。Java 堆不需要连续内存，并且可以通过动态增加其内存，增加失败会抛出 OutOfMemoryError 异常。(下图是基于jdk1.7版本，jdk1.8永久代变换成元空间Metaspace)



1.2 垃圾回收

从年轻代空间（包括 Eden 和 Survivor 区域）回收内存被称为 Minor GC，从年老代空间回收内存被称为 Major GC，Full GC 是清理整个堆空间—包括年轻代和老年代。这使得我们不用去关心到底是叫 Major GC 还是 Full GC，大家应该关注当前的 GC 是否停止了所有应用程序的线程，还是能够并发的处理而不用停掉应用程序的线程。（Stop-the-world STW）

在 JVM 中，具体实现有 串行收集器(Serial)、新生代并行收集器(ParNew)、并行回收(Parallel Scavenge)、CMS(Concurrent Mark Sweep)、Serial Old（Serial收集器的老年代版本）、Parallel Old(Parallel Scavenge收集器的老年代版本)、G1 等。具体的使用原则在后面的JVM调优原则详细介绍。

2 调优层次

性能调优包含多个层次，比如：架构调优、代码调优、JVM调优、数据库调优、操作系统调优等。架构调优和代码调优是JVM调优的基础，其中架构调优是对系统影响最大的。

3 调优指标

3.1 吞吐量

重要指标之一，吞吐量是衡量系统在单位时间里面完成的工作数量。吞吐量需求通常忽略延迟或者响应时间。通常情况下，提升吞吐量需要以系统响应变慢和更多内存消耗作为代价。

一个吞吐量的例子：“这个应用需要每秒完成2500个事务。”

3.2 延迟或响应时间

延迟或者响应时间是衡量应用从接收到一个任务到完成这个任务消耗的时间。一个延迟或者响应时间的需求需要忽略吞吐量。通常来讲，提升应用的响应时间需要以更低吞吐量或提高应用的内容消耗。

一个延迟或者响应时间的例子:"例如系统处理一个HTTP请求需要200ms，这个200ms就是系统的响应时间。"

3.3 内存占用

内存占用是衡量应用消耗的内存，这个内存占用是指应用在运行在某一个吞吐量、延迟以及可用性和易管理性指标下的内存消耗，内存占用是通常描述为应用运行的时候Java堆的大小或者总共需要消耗内存。通常情况下，通过增加Java堆的大小以增加应用内存占用可以提升吞吐量或者减少延迟，或者两者兼具。当应用可用的内存减少的时候，吞吐量和延迟通常会受到损失。在给定内存的情况下，应用占用的内存可以限制应用的实例数（这个会影响可用性）。

一个例子说明内存占用的需求是：“这个应用会单独运行在一个8G的系统上面或者多出3个应用实例运行在一个24G的应用系统上面。”

4 何时JVM调优

遇到以下情况，就需要考虑进行JVM调优：

- 系统吞吐量与响应性能不高或下降；
- Heap内存（老年代）持续上涨达到设置的最大内存值；
- Full GC 次数频繁；
- GC 停顿时间过长（超过1秒）；
- 应用出现OutOfMemory 等内存异常；
- 应用中有使用本地缓存且占用大量内存空间；

5 JVM调优目标

调优的最终目的都是为了应用程序使用最小的硬件消耗来承载更大的吞吐量。jvm调优主要是针对垃圾收集器的收集性能优化，减少GC的频率和Full GC的次数，令运行在虚拟机上的应用能够使用更少的内存以及延迟获取更大的吞吐量和减少暂停时间。

下面展示了一些JVM调优的量化目标参考实例，注意：不同应用的JVM调优量化目标是不一样的。

- 堆内存使用率 <= 70%;
- 老年代内存使用率<= 70%;
- avgpause <= 1秒;
- Full GC 次数0 或 avg pause interval >= 24小时 ;

6 JVM调优原则

6.1 优先原则

优先架构调优和代码调优，JVM优化是不得已的手段，大多数的Java应用不需要进行JVM优化

6.2 堆设置

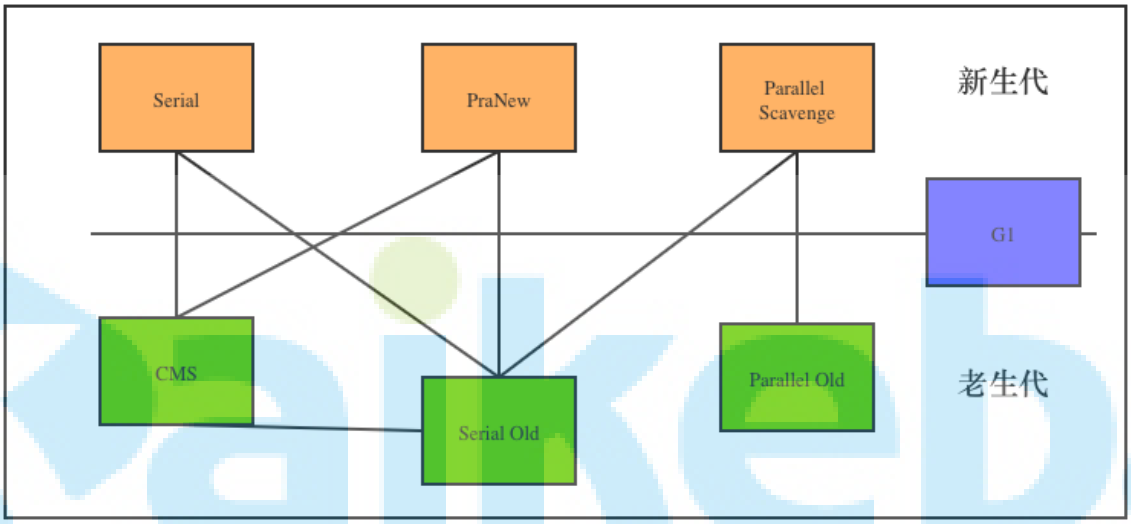
参数-Xms和-Xmx，通常设置为相同的值，避免运行时要不断扩展JVM内存，建议扩大至3-4倍FullGC后的老年代空间占用。

6.3 垃圾回收器设置

有 7 种不同的垃圾回收器，它们分别用于不同分代的垃圾回收。

- 新生代回收器：Serial、ParNew、Parallel Scavenge
- 老年代回收器：Serial Old、Parallel Old、CMS
- 整堆回收器：G1

  两个垃圾回收器之间有连线表示它们可以搭配使用，可选的搭配方案如下：



新生代	老年代
Serial	Serial Old
Serial	CMS
ParNew	Serial Old
ParNew	CMS
Parallel Scavenge	Serial Old
Parallel Scavenge	Parallel Old
G1	G1

6.4 年轻代设置

参数-Xmn，1-1.5倍FullGC之后的老年代空间占用。

避免新生代设置过小，当新生代设置过小时，会产生两种比较明显的现象，一是minor GC次数频繁，二是可能导致 minor GC对象直接进入老年代。当老年代内存不足时，会触发Full GC。

避免新生代设置过大，当新生代设置过大时，会带来两个问题：一是老年大变小，可能导致Full GC频繁执行；二是 minor GC 执行回收的时间大幅度增加。

6.5 年老代设置

响应时间优先的应用：年老代使用并发收集器，所以其大小需要小心设置，一般要考虑并发会话率和会话持续时间等一些参数。如果堆设置小了，可以会造成内存碎片、高回收频率以及应用暂停而使用传统的标记清除方式；如果堆大了，则需要较长的收集时间。

吞吐量优先的应用：一般吞吐量优先的应用都有一个很大的年轻代和一个较小的年老代。原因是，这样可以尽可能回收掉大部分短期对象，减少中期的对象，而年老代尽存放长期存活对象

6.6 永久代设置

基于jdk1.7版本，参数-XX:PermSize和-XX:MaxPermSize，基于jdk1.8版本，参数 -XX:MetaspaceSize和-XX:MaxMetaspaceSize，通常设置为相同的值，避免运行时要不断扩展，建议扩大至1.2-1.5倍FullGc后的永久带空间占用。

7 JVM调优步骤

7.1 监控分析

分析GC日志及dump文件，判断是否需要优化，确定瓶颈问题点。监控JVM的状况，可以使用阿里新开源的arthas，或者jconsole等工具查看运行状态。更多工具介绍将在后面详细介绍。

如何生成GC日志

开启GC日志，多种方法都能开启GC的日志功能，其中包括：使用-verbose:gc或-XX:+PrintGC这两个标志中的任意一个能创建基本的GC日志（这两个日志标志实际上互为别名，默认情况下的GC日志功能是关闭的）使用-XX:+PrintGCDetails标志会创建更详细的GC日志，推荐使用-XX:+PrintGCDetails标志（这个标志默认情况下也是关闭的）；通常情况下使用基本的GC日志很难诊断垃圾回收时发生的问题。

如何产生dump文件

1.JVM的配置文件中配置

JVM启动时增加两个参数:

```
#出现 OOME 时生成堆 dump:
-XX:+HeapDumpOnOutOfMemoryError

#生成堆文件地址:
-XX:HeapDumpPath=/home/hadoop/dump/
```

2.jmap生成

发现程序异常前通过执行指令，直接生成当前JVM的dump文件，9257是指JVM的进程号

jmap -dump:file=文件名.dump [pid]

```
jmap -dump:format=b,file=dump.dat 9257
```

由于第一种方式是一种事后方式，需要等待当前JVM出现问题后才能生成dump文件，实时性不高，第二种方式在执行时，JVM是暂停服务的，所以对线上的运行会产生影响。所以建议第一种方式。

7.2 判断

如果各项参数设置合理，系统没有超时日志出现，GC频率不高，GC耗时不高，那么没有必要进行GC优化，如果GC时间超过1-3秒，或者频繁GC，则必须优化。

7.3 确定目标

7.4 调整参数

调优一般是从满足程序的内存使用需求开始的，之后是时间延迟的要求，最后才是吞吐量的要求，要基于这个步骤来不断优化，每一个步骤都是进行下一步的基础，不可逆行之。具体的参数我们在后面详细介绍。

7.5 对比调优前后指标差异

7.6 重复以上过程

7.7 应用

找到最合适的参数，将这些参数应用到所有服务器，并进行后续跟踪。