JDK1.8 默认使用的Paraller GC + Paraller Old GC

java -XX:+PrintFlagsFinal 可以看到1.8默认的是 UseParallelGC ParallelGC 默认的是 Parallel Scavenge（新生代）+ Parallel Old（老年代）

在JVM中是+XX配置实现的搭配组合：

1. UseSerialGC 表示 “Serial” + "Serial Old"组合
2. UseParNewGC 表示 “ParNew” + “Serial Old”
3. UseConcMarkSweepGC 表示 “ParNew” + “CMS”. 组合，“CMS”是针对旧生代使用最多的
4. UseParallelGC 表示 “Parallel Scavenge” + "Parallel Old"组合
5. UseParallelOldGC 表示 “Parallel Scavenge” + "Parallel Old"组合

在实践中使用UseConcMarkSweepGC 表示 “ParNew” + “CMS” 的组合是经常使用的

FUll GC 的原因：

1. 老年代空间不足
2. 永久代空间不足
3. concurrent mode failure, 有对象要放入老年代，但老年代空间不足
4. promotion failed 晋升到老年代，但空间不足
5. 统计得到的Minor GC晋升到老年代的对象平均大小大于老年代剩余空间大小

GC调优的步骤：

1. 监控GC状态
2. 分析监控结果后决定是否需要优化GC. 比如GC时间只有0.1-0.3秒，那么就不需要把时间浪费在GC优化上，但如果运行的时间达到了1-3秒，甚至大于10s,那么GC优化将是很有必要的。
3. 设置合适的GC类型和内存大小
4. 分析结果 设置完GC参数后，分析24小时的结果，看看GC的情况，根据这个情况来具体的调整内存大小和GC类型。

频繁GC的原因：

1. 人为原因 比如在代码里调用System.GC ， Runtime.GC
2. 框架原因 一些框架可能内部调用GC方法
3. 内存的原因 当heap 大小设置的比较小时，会频繁的引发GC, 比如Spark对内存性能要求是比较高的，分配大的内存，可以显著减少频繁GC的发生。
4. 其它原因 对象的生命周期比较短的情况下，创建和释放是比较频繁的。

Minor GC ， Full GC 触发条件

- Minor GC触发条件：当Eden区满时，触发Minor GC。新创建的对象大小 > Eden所剩空间

- Full GC触发条件： 清理整个堆空间，包括年轻代和老年代 当年老代满时会引发Full GC，Full GC将会同时回收年轻代、年老代

当永久代满时也会引发Full GC，会导致Class、Method元信息的卸载

- （1）调用System.gc时，系统建议执行Full GC，但是不必然执行
- （2）老年代空间不足
- （3）方法去空间不足
- （4）通过Minor GC后进入老年代的平均大小大于老年代的可用内存
- （5）由Eden区、From Space区向To Space区复制时，对象大小大于To Space可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小。

CMS收集器

对于CMS收集器来说，最重要的是合理地设置年轻代和年老代的大小。年轻代太小的话，会导致频繁的Minor GC，并且很有可能存活期短的对象也不能被回收，GC的效率就不高。而年老代太小的话，容纳不下从年轻代过来的新对象，会频繁触发单线程Full GC，导致较长时间的GC暂停，影响Web应用的响应时间。

G1收集器

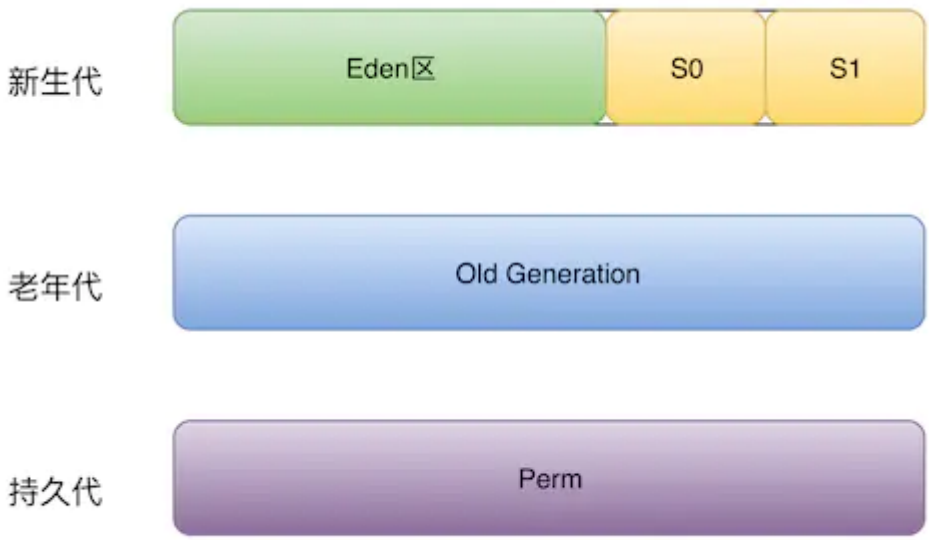
对于G1收集器来说，不推荐直接设置年轻代的大小，这一点跟CMS收集器不一样，这是因为G1收集器会根据算法动态决定年轻代和年老代的大小。因此对于G1收集器，需要关心的是Java堆的总大小（-Xmx）。

此外G1还有一个较关键的参数是-XX:MaxGCPauseMillis = n，这个参数是用来限制最大的GC暂停时间，目的是尽量不影响请求处理的响应时间。G1将根据先前收集的信息以及检测到的垃圾量，估计它可以立即收集的最大区域数量，从而尽量保证GC时间不会超出这个限制。因此G1相对来说更加“智能”，使用起来更加简单。

。

GC 优化

优化并不能解决一切问题，是最后的调优手段



- 新生代： 每次回收叫Minor GC，后只要少量存活对象，选用复制算法，只需要复制少量就可完成
- 老年代，回收叫Major Fc, 对象存活率比较高
- Full GC （回收新， 老）
- 永久代，存放元数据，如class, Method的元信息，与垃圾回收关系不大

GC 的概念

从JDK7 起，有5种GC类型

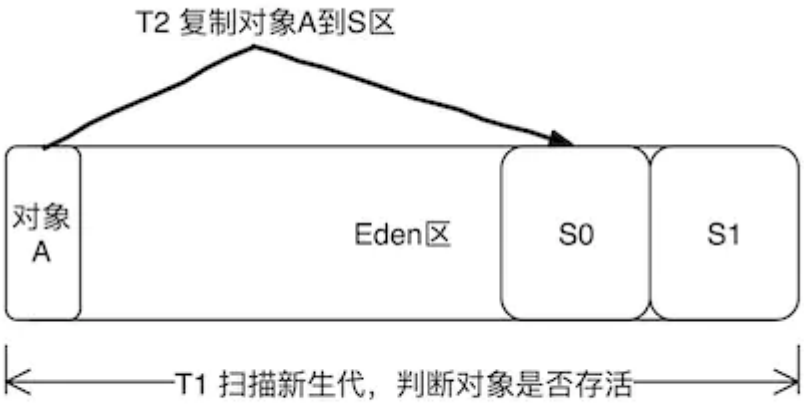
1. CMS (concurrent mark & Sweep)
2. G1 (Garbage First)
3. Parallel GC （多线程，速度快，内存很大时很有用）
4. Parallerl Old GC
5. Serial GC （单线程， 专为单核计算机设计，适合小内存， 每次GC 都会把堆分两部分，一部分有对象，一部分没对象）

优化概念

JVM 为了执行GC将会暂停运行中的应用，除GC线程外，应用中的其它所有线程都将暂停工作。优化一般指减少STW的时间

优化方法

- Minor GC频繁： 考虑增大新生代空间



复制对象的成本远高于扫描成本，单次Minor GC时间更多取决于GC后存活对象的数量，而非Eden区的大小。

> 新生代扩容后老年代增速也会变慢，Major GC 频率也会降低

> PS: 如果有大量短期对象，应该选择较大的年轻代， 如果有较多的持久对象，应该适当加大老年代

- 减少CMS Remark 时间 CMS在remark 时会扫描新生代和老年代，所以减少新生代对象数量即可（如CMS执行前强行进行一次Minor GC）
- 避免Minor GC扫描全表 卡表的引入：
- 老年代分成大小512b的若干卡表，当发生老年代引用新生代对象时，虚拟机将该卡对应的卡表元素设置为适当的值。 这样Minor Gc时就可以知道那些在老年代中存在引用
- 避免Full GC 如提示失败，concurrent model failure 时候等由于老年代空间不足导致的失败

参考文献

- 《深入理解Java虚拟机——JVM高级特性与最佳实践》 - 周志明

- 简书
- 掘金