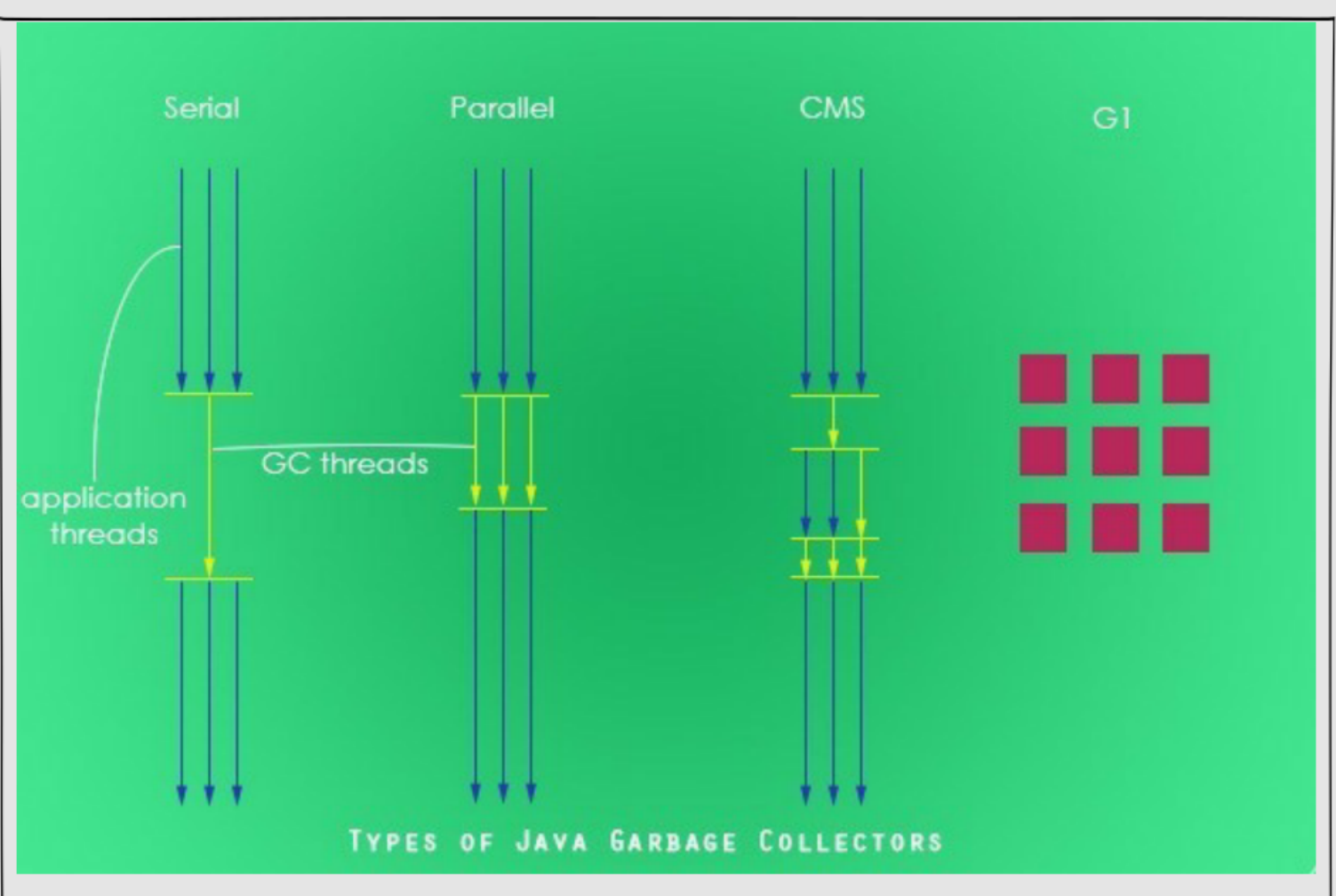
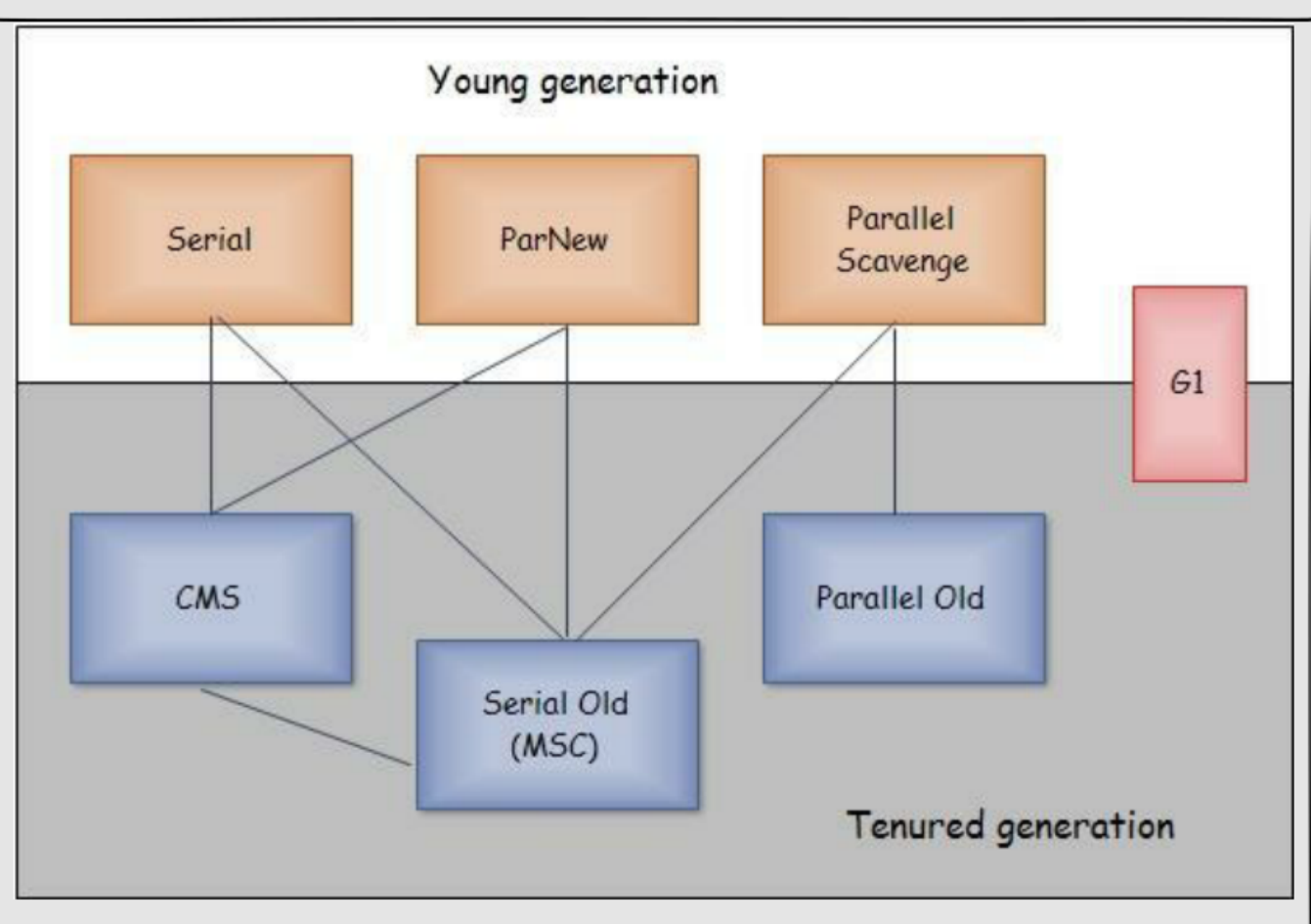
JVM

- 思路
  - 运行时数据区
  - 堆
    - 分代
      - 新生代
        - 8:1:1 为什么？
          - 98%能回收
          - 不能98%回收
            - 分配担保
      - 老年代
      - 永久代\Meta Space 1.8
      - 各个分代大小多少合适
      - 新生代晋升老年代
        - 对象优先分配在Eden
          - -XX:SurvivorRatio=8
          - -Xms20M
          - -Xmx
          - -Xmn
          - -XX:+PrintGCDetails
        - 对象分配TLAB
          - 栈上分配
          - 逃逸分析
          - 指针碰撞
            - cas同步
          - 空闲列表
        - 大对象直接进入老年代
          - -XX:PretenureSizeThreshold=3145728=3M
        - 长期存活的对象将进入老年代
          - age
          - -XX:MaxTenuringThreshold=1
        - 动态对象年龄判定
          - 相同年龄所有对象的大小总和 > Survivor空间的一半
            - 直接晋级 不需要判定 MaxTenuringThreshold
        - 空间分配担保
          - Minor GC 之前检查 老年代最大可用连续空间是否>新生代所有对象总空间
            - 大于
              - Minor GC
            - 不大于
              - 老年代最大可用连续空间是否大于历次晋升到老年代对象的平均大小
                - 大于
                  - Minor GC
                - 不大于
                  - Full GC
                - -XX：-HandlePromotionFailure (1.6Update 24之后不生效)
                  - 允许担保失败
                    - 大于
                      - Minor GC
                    - 不大于
                      - Full GC
                  - 不允许担保失败
                    - Full GC
    - 垃圾回收
      - 什么样的对象需要被回收
        - 判断算法
          - 引用计数法
          - 可达性分析算法
            - GC Roots
              - 虚拟机栈中本地变量表引用的对象
              - 方法区中
                - 类静态变量引用的对象
                - 常量引用的对象
              - 本地方法栈中JNI引用的对象
            - 不可达了还会发生什么
              - finalize()
        - 引用
          - 强引用
          - 软引用
            - 内存不足时回收
              - 缓存
          - 弱引用
            - 下一次垃圾收集会回收掉，不管内存是否足够
              - 大对象
          - 虚引用
      - 回收
        - 垃圾回收算法-方法论
          - 标记-清除算法
            - pros
              - first
            - cons
              - 效率不高
              - 空间碎片
          - 复制算法
            - pros
              - 实现简单、高效、不用考虑碎片
            - cons
              - 要分一块出来做复制倒
          - 标记-整理算法
        - 垃圾回收器-hotspot实现
          - Serial
            - 历史悠久
            - 单线程
            - client
          - ParNew
            - 多线程
              - 多核下对资源的利用率会高
            - -XX:+ UseConcMarkSweepGC 的默认新生代收集器
            - -XX：ParallelGCThreads
              - GC线程数
          - Parallel Scavenge
            - 吞吐量
              - 吞吐量 = 运行用户代码时间 / （运行用户代码时间 + 垃圾收集时间）
            - -XX:MaxGCPauseMillis
              - 垃圾回收时间尽量不超过
            - -XX:GCTimeRatio
              - 垃圾收集时间占总时间的比率
            - -XX:UseAdaptiveSizePolicy
              - 自动设置 动态调整
              - GC Ergonomics
          - Serial Old
            - client
            - server
              - 与Paralle Scavenge
              - CMS的后备预案
                - Concurrent Mode Failusre时使用
          - Parallel Old
            - 标记-整理算法
          - CMS Concurrent Mark Sweep
            - 标记-清除算法
            - 减少回收停顿时间
            - 步骤
              - 初始标记
                - 快
                - 标记GC Roots能直接关联的对象
              - 并发标记
                - GC Roots Tracing
              - 重新标记
                - 修正并发标记期间用户程序继续运行产生变动的那一部分对象的标记记录
                - 比初始标记时间长
                - 比并发标记时间短
              - 并发清除
                - 清除而不是整理 意味着有碎片
            - 缺点 cons
              - CPU敏感
                - GC线程占用CPU资源导致 吞吐量下降
              - 无法处理浮动垃圾
                - 垃圾收集阶段用户程序还要运行所以还要预留空间 不能被填满再回收
                  - -XX:CMSInitiationgOccupancyFraction
                    - Concurrent Mode Failure 启用Serial Old
              - 碎片
                - -XX:+UseCMSCompactAtFullCollection
                  - FullGC时开启内存碎片的合并整理过程
                - -XX:CMSFullGCsBeforeCompaction
                  - 执行多少次不压缩FullGC后 来一次带压缩的 0 表示每次都压
            - -XX:+UseConcMarkSweep
            - +XX:+ScavengeBeforeRemark 触发Minor GC



http://incdn1.b0.upaiyun.com/2014/11/fcc8b26b654194d19118c91c0a73f432.jpg

          - G1
            - G1垃圾回收器适用于堆内存很大的情况，他将堆内存分割成不同的区域，并且并发的对其进行垃圾回收。
            - G1也可以在回收内存之后对剩余的堆内存空间进行压缩。
            - 并发扫描标记垃圾回收器在STW情况下压缩内存。
            - G1垃圾回收会优先选择第一块垃圾最多的区域
            - 通过JVM参数 -XX:+UseG1GC 使用G1垃圾回收器
            - http://ifeve.com/%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3g1%E5%9E%83%E5%9C%BE%E6%94%B6%E9%9B%86%E5%99%A8/



组合

        - 什么时间节点回收
          - GC Roots枚举效率
            - OopMap Ordinary Object Pointer
          - 安全点
            - 方法调用
            - 循环跳转
            - 异常跳转
            - 抢先式中断
              - 先暂停 没在安全点的继续跑
            - 主动式中断
              - 设置GC 标志
                - 看到标志就停
          - 安全域
            - 线程挂起怎么办
            - 一段代码中，引用关系不会发生变化
              - 1.标识线程自己进入了safe regin
              - 2.离开时检查系统是否已经完成GC过程
          - -XX:+PrintGCApplicationStoppedTime