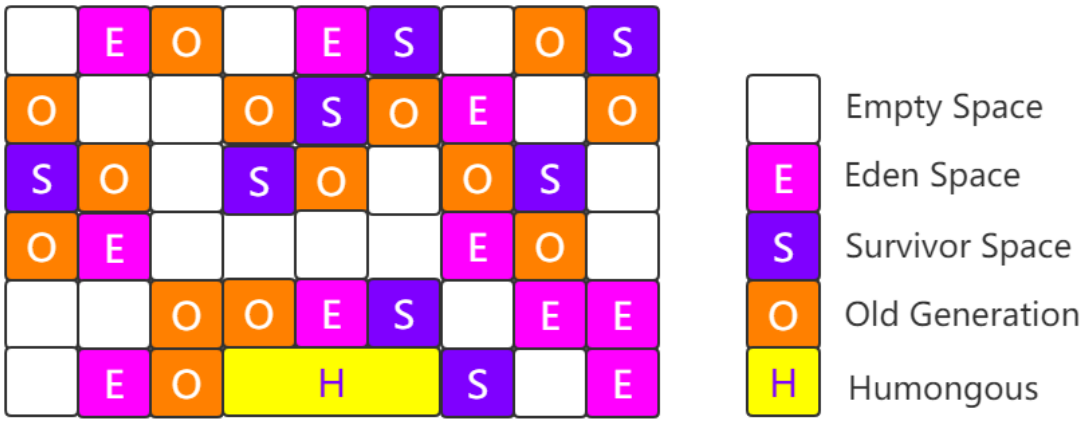
G1(Garbage-First)

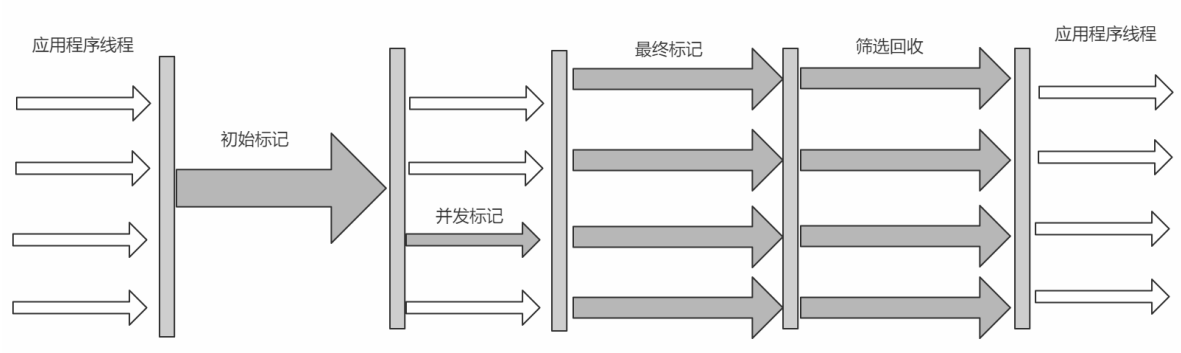
> 官网： https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/g1_gc.html#garbage_first_garbage_collection
>
> 使用G1收集器时，Java堆的内存布局与就与其他收集器有很大差别，它将整个Java堆划分为多个大小相等的独立区域（Region），虽然还保留有新生代和老年代的概念，但新生代和老年代不再是物理隔离的了，它们都是一部分Region（不需要连续）的集合。
>
> 每个Region大小都是一样的，可以是1M到32M之间的数值，但是必须保证是2的n次幂
>
> 如果对象太大，一个Region放不下[超过Region大小的50%]，那么就会直接放到H中
>
> 设置Region大小：-XX:G1HeapRegionSize=M
>
> 所谓Garbage-Frist，其实就是优先回收垃圾最多的Region区域

> （1）分代收集（仍然保留了分代的概念）
> （2）空间整合（整体上属于“标记-整理”算法，不会导致空间碎片）
> （3）可预测的停顿（比CMS更先进的地方在于能让使用者明确指定一个长度为M毫秒的时间片段内，消耗在垃圾收集上的时间不得超过N毫秒）

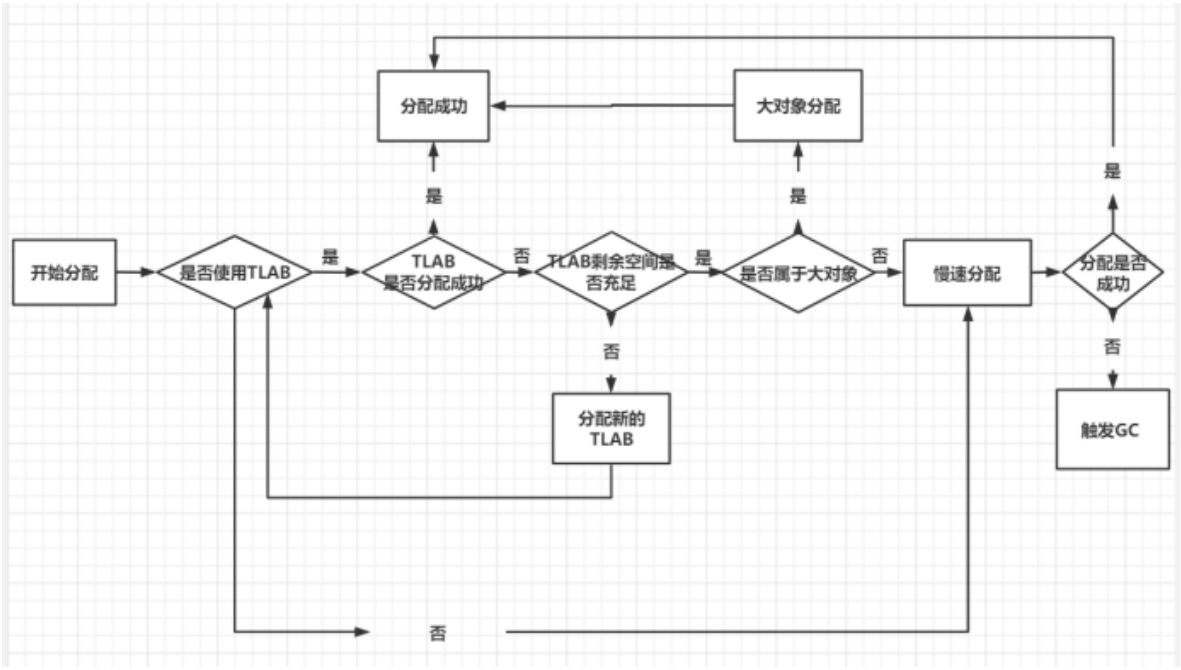


工作过程可以分为如下几步

初始标记（Initial Marking）	标记以下GC Roots能够关联的对象，并且修改TAMS的值，需要暂停用户线程
并发标记（Concurrent Marking）	从GC Roots进行可达性分析，找出存活的对象，与用户线程并发执行
最终标记（Final Marking）	修正在并发标记阶段因为用户程序的并发执行导致变动的数据，需暂停用户线程
筛选回收（Live Data Counting and Evacuation）	对各个Region的回收价值和成本进行排序，根据用户所期望的GC停顿时间制定回收计划



TLAB流程



相关参数

> -XX: +UseG1GC 开启G1垃圾收集器
>
> -XX: G1HeapReginSize 设置每个Region的大小，是2的幂次，1MB-32MB之间
>
> -XX:MaxGCPauseMillis 最大停顿时间
>
> -XX:ParallelGCThread 并行GC工作的线程数
>
> -XX:ConcGCThreads 并发标记的线程数
>
> -XX:InitiatingHeapOcccupancyPercent 默认45%，代表GC堆占用达到多少的时候开始垃圾收集

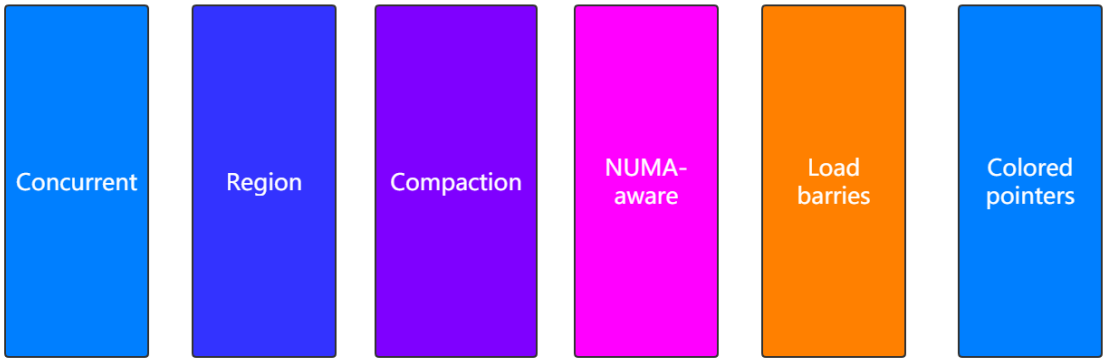
2.5.5.8 ZGC

> 官网： https://docs.oracle.com/en/java/javase/11/gctuning/z-garbage-collector1.html#GUID-A5A42691-095E-47BA-B6DC-FB4E5FAA43D0
>
> JDK11新引入的ZGC收集器，不管是物理上还是逻辑上，ZGC中已经不存在新老年代的概念了
>
> 会分为一个个page，当进行GC操作时会对page进行压缩，因此没有碎片问题
>
> 只能在64位的linux上使用，目前用得还比较少

（1）可以达到10ms以内的停顿时间要求

（2）支持TB级别的内存

（3）堆内存变大后停顿时间还是在10ms以内



2.5.5.9 垃圾收集器分类

- **串行收集器**->Serial和Serial Old

只能有一个垃圾回收线程执行，用户线程暂停。

`适用于内存比较小的嵌入式设备`。

- **并行收集器**[吞吐量优先]->Parallel Scanvenge、Parallel Old

多条垃圾收集线程并行工作，但此时用户线程仍然处于等待状态。

`适用于科学计算、后台处理等若交互场景`。

- **并发收集器**[停顿时间优先]->CMS、G1

用户线程和垃圾收集线程同时执行(但并不一定是并行的，可能是交替执行的)，垃圾收集线程在执行的时候不会停顿用户线程的运行。

`适用于相对时间有要求的场景，比如web`。

2.5.5.10 常见问题

- 吞吐量和停顿时间
  - 停顿时间->垃圾收集器 `进行` 垃圾回收终端应用执行响应的时间
  - 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)

  > 停顿时间越短就越适合需要和用户交互的程序，良好的响应速度能提升用户体验；
  > 高吞吐量则可以高效地利用CPU时间，尽快完成程序的运算任务，主要适合在后台运算而不需要太多交互的任务。

  `小结`:这两个指标也是评价垃圾回收器好处的标准。
- 如何选择合适的垃圾收集器

  > https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/collectors.html#sthref28

  - 优先调整堆的大小让服务器自己来选择
  - 如果内存小于100M，使用串行收集器
  - 如果是单核，并且没有停顿时间要求，使用串行或JVM自己选
  - 如果允许停顿时间超过1秒，选择并行或JVM自己选
  - 如果响应时间最重要，并且不能超过1秒，使用并发收集器
- 对于G1收集

JDK 7开始使用，JDK 8非常成熟，JDK 9默认的垃圾收集器，适用于新老生代。

是否使用G1收集器?

```
（1）50%以上的堆被存活对象占用
（2）对象分配和晋升的速度变化非常大
（3）垃圾回收时间比较长
```

- G1中的RSet

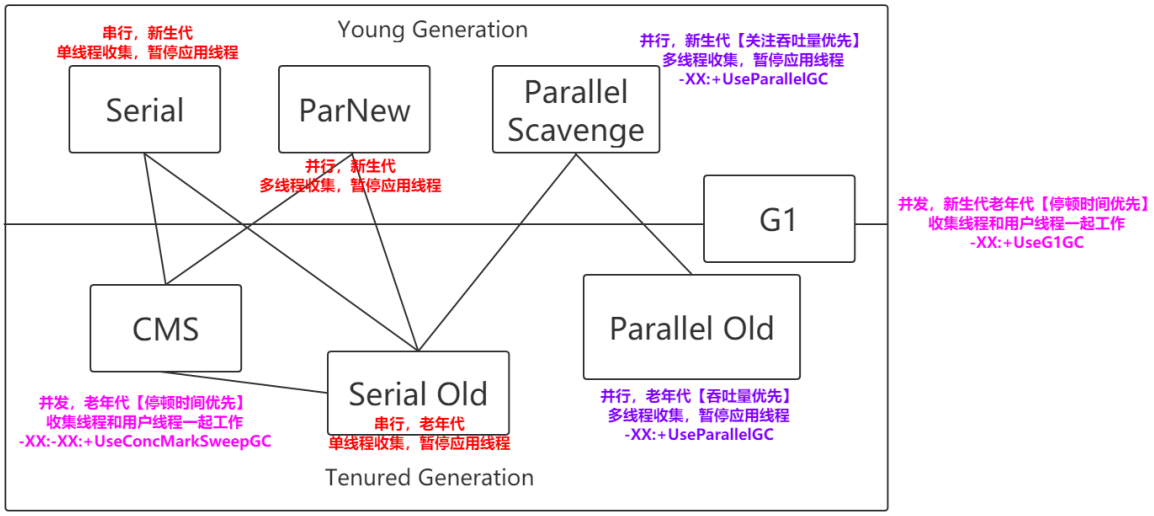
全称Remembered Set，记录维护Region中对象的引用关系

```
试想，在G1垃圾收集器进行新生代的垃圾收集时，也就是Minor GC，假如该对象被老年代的Region中所引用，这时候新生代的该对象就不能被回收，怎么记录呢？
不妨这样，用一个类似于hash的结构，key记录region的地址，value表示引用该对象的集合，这样就能知道该对象被哪些老年代的对象所引用，从而不能回收。
```

- 如何开启需要的垃圾收集器

> 这里JVM参数信息的设置大家先不用关心，后面会学习到。

```
（1）串行
    -XX：+UseSerialGC
    -XX：+UseSerialOldGC
（2）并行(吞吐量优先)：
    -XX：+UseParallelGC
    -XX：+UseParallelOldGC
（3）并发收集器(响应时间优先)
    -XX：+UseConcMarkSweepGC
    -XX：+UseG1GC
```



源码部分我就不贴了，因为确实不好贴。