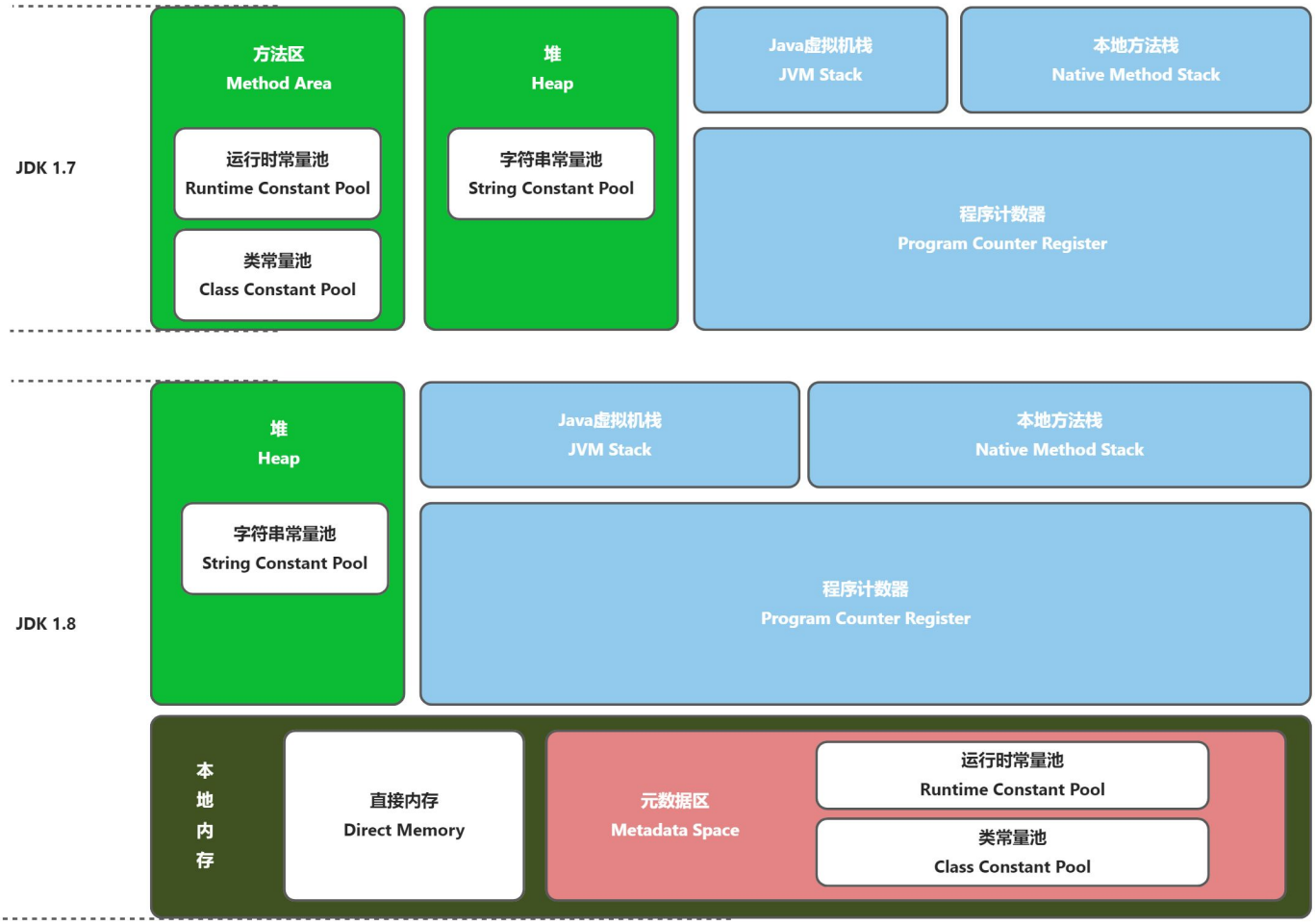
JVM浅析

java运行时数据区



方法区

- 存放：类的信息，静态变量，final常量，Field信息，方法信息等
- 特点：线程共享，一定条件会被GC，会抛OOM异常

方法区类似于一种规范，JDK1.8之前，方法区在HotSpot VM的实现是永久代，JDK1.8之后的实现元数据区

方法区中包含运行时常量池和类常量池两部分

虚拟机栈

- 存放：局部变量表，操作栈，动态链接，方法出口信息等
- 特点：线程私有，一定条件下会抛StackOverFlowError和OutOfMemoryError

方法被线程执行就是创建栈帧，方法的调用开始和结束即是栈帧入栈和出栈的过程

扩展：局部变量表存放八大原始数据类型，以及对象引用和returnAddress,局部变量表所需的内存空间在编译期间完成分配，并且在方法运行期间不改变其大小。

堆

- 存放：java对象实例，数组
- 特点：线程共享，是GC的主要区域，会出现OOM异常

堆主要由年轻代和老年代组成，年轻代又可以分为eden,s0和s1三个区域

问：为什么要堆进行分代

答：对象创建时存放在新生代Eden区，经过多次Young GC之后，对象会进入老年代，所以新生代内存相对较小，大多数对象都朝生夕死，GC会比较频繁，而老年代内存相对较大，对象生命周期较长，GC没有那么频繁，然后我们可以根据新生代和老年代对象的特点选择合适的垃圾收集算法，更高效率进行垃圾回收

本地方法栈

- 存放：局部变量表，操作栈，动态链接，方法出口信息等
- 特点：线程独享，一定条件下会抛StackOverFlowError和OutOfMemoryError

区别于虚拟机栈，本地方法栈调用的是native方法，底层是C实现的

程序计数器

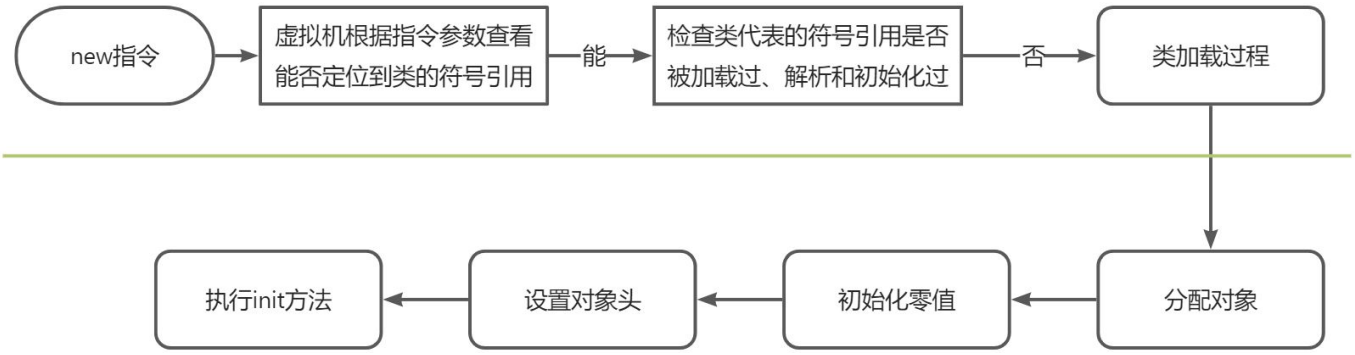
作用：用来表示当前线程所执行的字节码的行号指示器

特点：线程私有，唯一不会发生OOM异常的区域

java对象生命周期

对象创建过程

类加载检查



1. 类加载检查
2. 内存分配

往下看

3. 初始化零值

分配内存完成后，虚拟机将分配到的内存空间都初始化为零值（不包括对象头），保证对象的实例字段在java程序中不赋初始值就能直接使用

4. 设置对象头

初始化零值后，设置对象头信息，如对象的哈希码，对象的GC分代年龄，是否启用偏向锁等，这些信息都是存放在对象头中的

5. 执行init方法

上述操作完成后，执行init方法，即设置代码中给对象赋的一些信息。

对象内存分配

对象所需内存大小在类加载完毕后已确定，虚拟机为对象分配内存即从堆中划分出一块固定大小的空间分配给new的对象，内存分配方式有两种；

- 指针碰撞

适用场合：堆内存规整即没有内存碎片的情况

原理：use内存和free内存分界，中间分界指针，分配内存时，指针向空闲内存移动相应的对象内存大小的空间

- 空闲列表

适用场合：堆内存不规整的情况

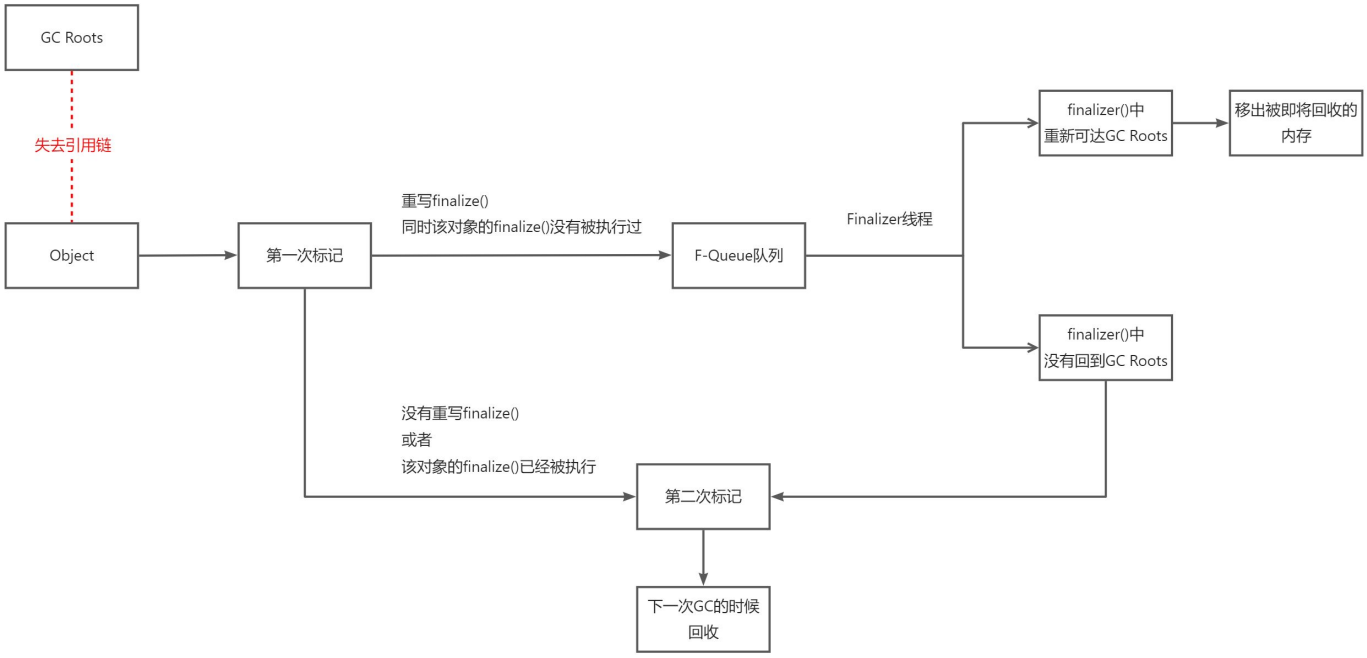
原理：虚拟机维护一个列表，列表记录可用的内存块，分配内存时，寻找足够大的内存块分配给对象实例，最后更新列表记录，如果列表内存空间不足则触发GC回收堆空间

堆内存分配并发问题：

- cas+失败重试策略
- TLAB策略

对象销毁过程

Java对象的销毁指的是释放对象占用的内存空间，JVM通过GC机制实现内存的自动回收



1. 新生代Eden区满时，发生Minor GC，通过GC Roots引用链找存活的对象，回收垃圾对象
2. 经过一次Minor GC，还存活的对象年龄+1，默认情况下对象年龄达到15会被移入老年代
3. 老年代内存满时，发生Major GC，回收老年代的垃圾对象
4. 经过Major GC还内存不足时，发生Full GC，同时回收新生代和老年代的垃圾对象
5. 经过Full GC还内存不足时，则抛Java.lang.OutOfMemoryError

对象访问方式

1. 句柄

特点：虚拟机栈本地变量表中的reference存储的是对象的句柄地址，句柄中包含对象实例数据与类型数据各自的地址信息

优点：当对象移动时，只需要更改句柄中实例数据的地址，reference不需要改动

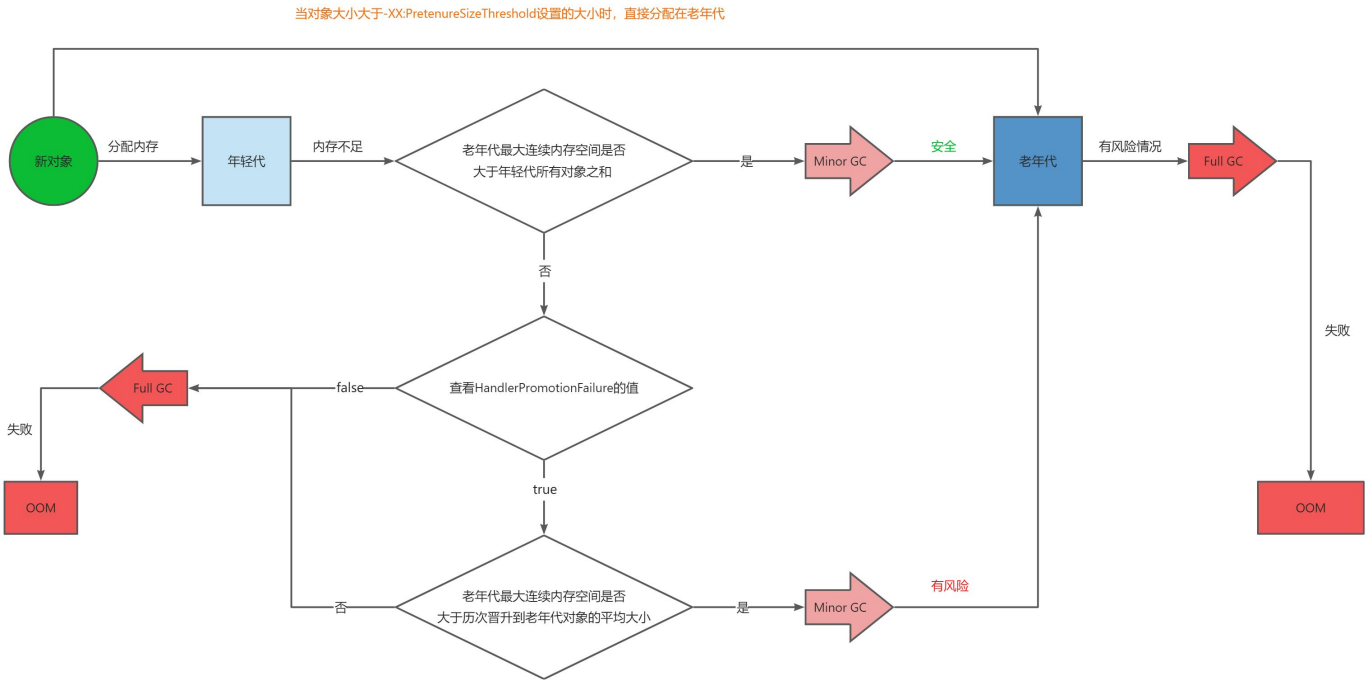
2. 直接指针

特点：reference存储的是对象的直接地址

优点：相对于使用句柄访问，直接指针节省了一次指针定位的时间开销，在java中对象访问十分频繁，所以累加起来可以节省不少的时间成本

为什么需要内存担保

what：新生代进行Minor GC前，判断老年代最大可用连续存储空间是否大于新生代所有对象之和，如果大于，则安全进行Minor GC，如果小于，则查看HandlerPromotionFailure的值，如果为false，则进行FullGC，如果为true,则继续判断老年代最大可用连续存储空间是否大于历次晋升到老年代的对象的平均大小，如果大于，则尝试进行Minor GC，此次的Minor GC是有风险的，如果小于，则进行Full GC



why：最大程度减少Full GC的次数，减少性能消耗

垃圾收集算法及特点

标记清除算法

原理：分为标记阶段和清除阶段，标记阶段从GC Roots开始扫描，标记所有引用链上的对象，未被标记的对象为垃圾对象，清除阶段把所有垃圾对象进行清除

适用场合：存活对象较多的情况下（老年代）

缺点：扫描了两次内存空间；容易产生内存碎片，可能造成资源浪费

标记复制算法

原理：经过标记阶段把所有存活的对象标记，然后开辟一块新的内存空间，把所有存活的对象复制到新的内存中，最后把当前内存所有对象清除

适用场合：存活对象较少的情况（新生代）

缺点：需要开辟新的内存空间，内存利用率不高

标记整理算法

原理：基于标记清除算法进行优化，在标记阶段完成后，将所有存活对象压缩到内存的一端，然后清理端边界外所有的对象

适用场合：存活对象较多的情况（老年代）

垃圾收集器及特点

Serial

特点：新生代垃圾回收器，单线程垃圾收集，采用标记复制算法，进行GC时，需要STW

优点：简单高效，没有线程交互的开销，适用于Client模式和单核服务器

参数设置：-XX:+UseSerialGC　　指定使用Serial垃圾回收器

Serial Old

Serial的老年代版本

特点：老年代垃圾回收器，单线程垃圾收集，采用标记整理算法，进行GC时，需要STW

优点：同Serial

ParNew

Serial的多线程版本

特点：新生代垃圾回收器，多线程垃圾收集，采用标记复制算法，进行GC时，需要STW

优点：并发进行垃圾收集，有效利用系统资源，减少STW的时间

参数设置：

- –XX:+UseConcMarkSweepGC　　指定使用CMS，默认使用ParNew作为新生代收集器
- –XX:+UseParNewGC　　强制指定ParNew
- –XX:ParallelGCThreads　　指定垃圾收集的线程数，ParNew默认开启线程数等于CPU核数

Parallel Scavenge

特点：新生代垃圾回收器，多线程垃圾回收，采用标记复制算法，进行GC时，需要STW，吞吐量可控制

优点：吞吐量优先，高效利用CPU，但是单个GC周期会变长，适用于后台运算，不需要太多用户交互的场景

参数设置：

- –XX:MaxGCPauseMillis　　最大垃圾收集停顿时间
- –XX:GCTimeRatio　　垃圾收集时间占总时间的比率，0<n<100的整数

Parallel Old

Parallel Scavenge的老年代版本

特点：老年代垃圾回收器，多线程垃圾收集，采用标记整理算法，进行GC时，需要STW

优点：同Parallel Scavenge

参数设置：–XX:UseParallelOldGC　　指定使用Parallel Old垃圾回收器

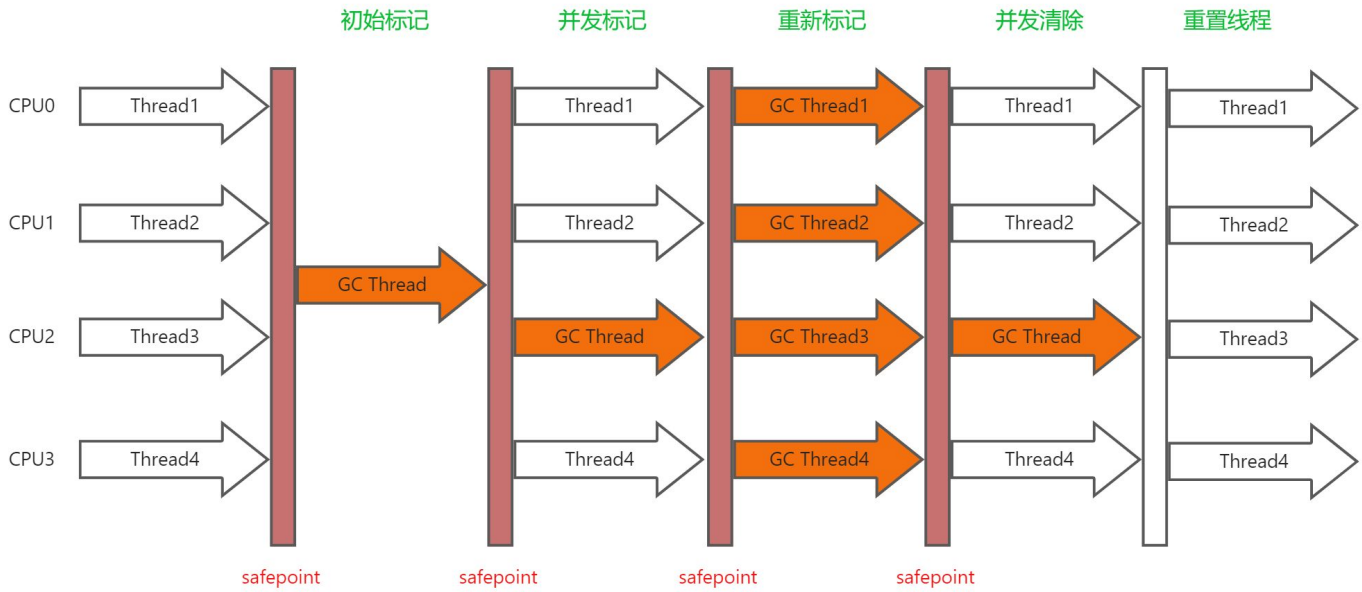
CMS(Concurrent Mark Sweep)

特点：老年代垃圾回收器，多线程垃圾收集，采用标记清除算法，以获取最短回收停顿时间为目标

优点：并发收集，低停顿，吞吐量大，适用于用户交互要求高，服务响应数据快的场景

缺点：对CPU资源敏感，无法收集浮动垃圾，容易产生大量内存碎片；适用于内存模型

流程图



- 初始标记：标记GC Roots能够直接关联到达的对象，会STW
- 并发标记：进行GC Root Tracing的过程，不会STW
- 重新标记：修正并发标记期间因用户程序运行而导致标记产生变动的那部分标记记录，会STW，时间较短
- 并发清除：用标记清除算法回收对象

参数设置：-XX:UseConcMarkSweepGC　　指定使用CMS回收器

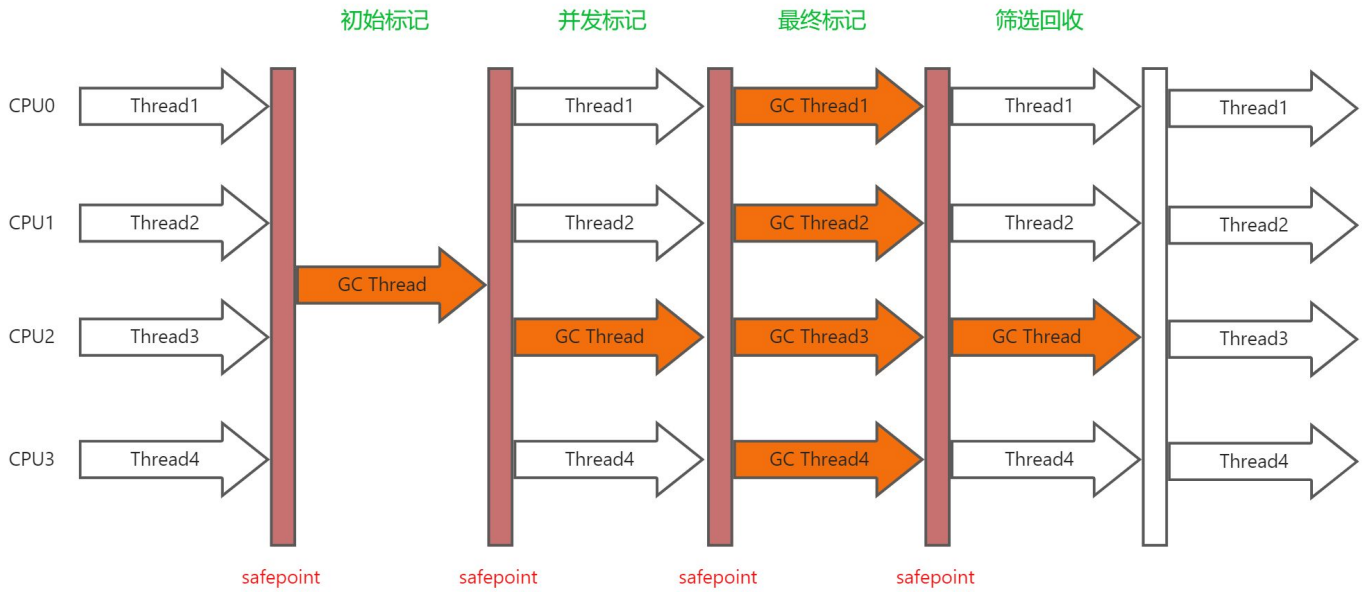
G1(Garbage-First)

特点：不区分新生代和老年代，而是将堆空间分为若干个Region，每个Region都包含逻辑上的年轻代和老年代；G1整体上采用标记整理算法，每个Region采用标记复制算法

优点：并行和并发进行垃圾回收，结合多种垃圾收集算法，空间整合，不产生内存碎片，有利于长时间运行；能独立管理整个堆空间，不需要与其他垃圾收集器搭配；停顿时间可预测，可以实现低停顿和高吞吐量；适用于堆内存较大的场景（6G~8G）

> 参考：https://blog.csdn.net/Hao_JunJie/article/details/124362872

流程图



- 初始标记：标记GC Roots能够直接关联到达的对象，耗时短的STW
- 并发标记：进行GC Root Tracing的过程，不会STW
- 最终标记：修正并发标记期间因用户程序运行而导致标记产生变动的那部分标记记录，会STW，时间较短
- 筛选回收：对每个Region的回收成本进行排序，按照用户自定义回收时间来制定回收计划

参数设置：

- –XX:UseG1GC 指定使用G1垃圾回收器
- –XX:+G1HeapRegionSize 设置每个Region大小，值为2的n次幂，范围为1~32M，默认是堆内存的1/2000，目的是根据最小的堆内存划分出约2048个区域
- –XX:MaxGCPauseMillis 设置期望最大停顿时间，默认200ms
- –XX:ParallelGCThread 设置STW期间的GC线程数，最大8个
- –XX:ConcGCThread 设置并发标记的线程数，约为ParallelGCThread的1/4
- –XX:InitiatingHeapOccupancyPercent 设置触发并发GC周期Java堆占用率的阈值，超过阈值触发GC，默认为45