

Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心



Java虚拟机垃圾回收机制

硕士研究生 宋言言

2017年12月10日

- 背景简介
- 基本概念
 - Java虚拟机运行时数据区
 - 垃圾收集 (Garbage Collection, GC)
- 垃圾收集过程
 - 对象已死吗
 - 垃圾收集算法
- 垃圾收集器

79A9虚拟机垃圾回收机制



背景简介

- Java是一门严谨的、面向对象的编程语言，同时是一个由一系列计算机软件和规范形成的技术体系。
- Java技术体系广泛应用于嵌入式系统、移动终端、企业服务器、大型机等各种场合，吸引了900多万软件开发者，这是全球最大的软件开发团队。



- C++：在内存管理领域，开发人员既是皇帝，又是劳动人民——既拥有一个对象的所有权，又担负着每一个对象生命开始到终结的维护责任。
- Java：基于**自动内存管理机制（内存分配和回收）**，不再需要为每一个new操作写delete/free代码，不容易出现内存泄漏和溢出问题。但一旦出现这些问题，则很难排查原因，因此了解虚拟机管理内存的机制是很必要的。

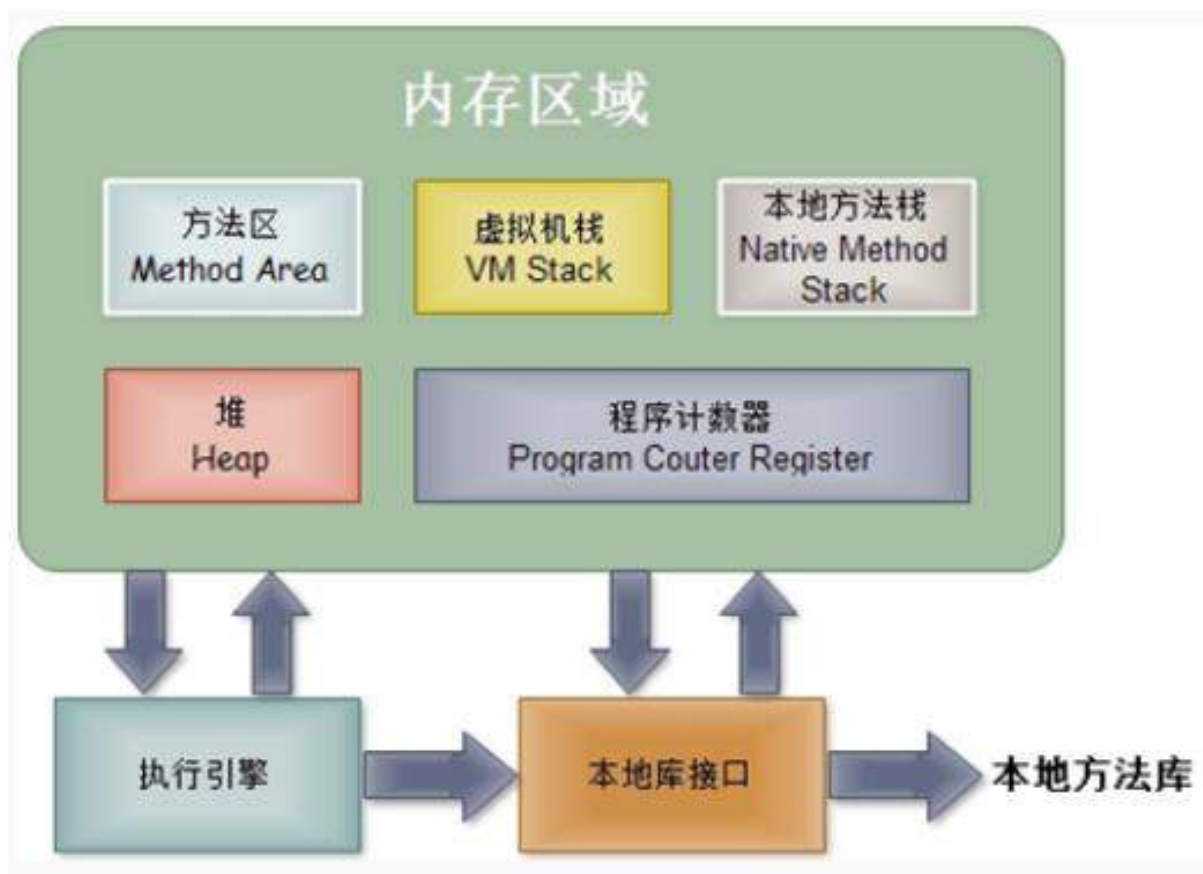
Java与C++之间有一堵由内存动态分配和垃圾收集技术所围成的“高墙”，墙外面的人想进去，墙里面的人想出来。





基本概念

- Java虚拟机运行时的数据区域



- Java虚拟机运行时各数据区域的功能

区域名称	存储数据类型	特点
程序计数器	当前线程所执行的字节码的行号指示器	线程私有
虚拟机栈	Java方法的局部变量表、操作数栈、动态链接、方法出口	线程私有
本地方法栈	native方法的局部变量表、操作数栈、动态链接、方法出口	线程私有
Java堆	对象实例和数组	线程共享
方法区	存储已被虚拟机加载的类信息、常量、静态变量、及时编译器编译后的代码等数据。	线程共享

- 垃圾收集区域
 - 程序计数器、虚拟机栈、本地方法栈三个区域随着线程而生，随线程灭而灭。
 - 因此垃圾收集主要针对java堆和方法区的内存。
- 垃圾收集数据类型
 - 堆内存中无用的对象
 - 方法区中废弃的常量和无用的类。

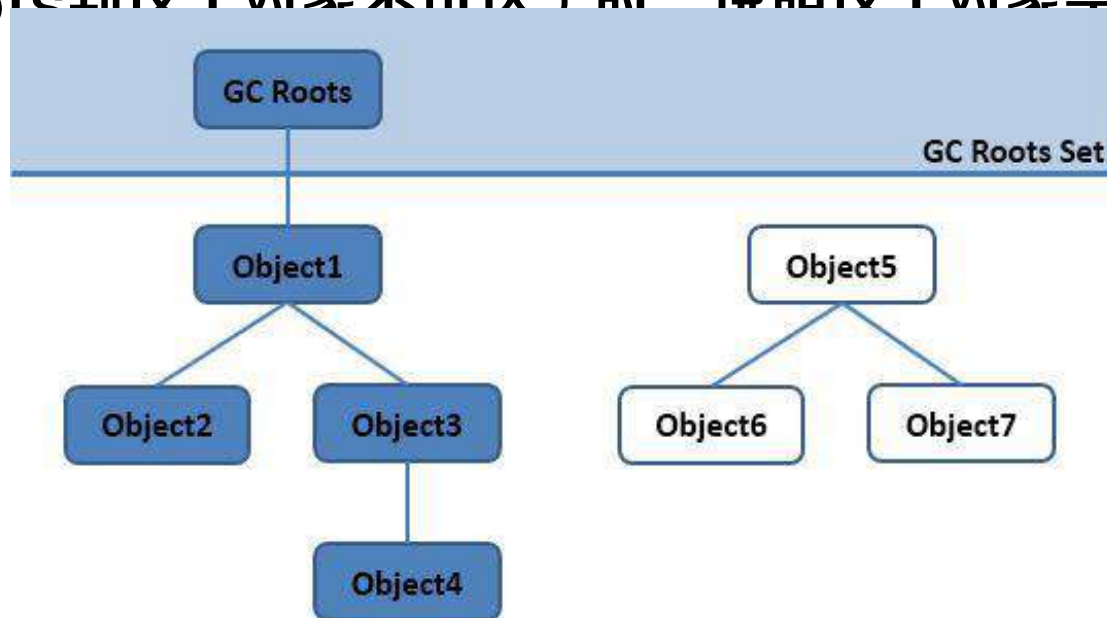


垃圾收集过程

- 引用计数算法
 - 给对象添加一个引用计数器，每当有一个地方引用他时，计数器值就加1；当引用失效时，计数器值减1。
 - 特点：实现简单，判定效率高，但是很难解决对象之间相互循环引用的问题，因此主流虚拟机里没有采用。



- 可达性分析法
 - 以一系列“GC Roots”对象为起始点，从这些节点开始向下搜索，搜索所走过的路径成为引用链（Reference Chain）。
 - 当一个对象到GC Roots没有任何引用链相连（就是从GC Roots到这个对象不可达）时，说明这个对象是不可用的。



- 哪些对象可以作为GC Roots节点呢？

虚拟机栈（栈帧中本地变量表）中引用的对象

方法区中静态属性引用的对象

本地方法栈中JNI引用的对象

方法区中常量引用的对象

- 引用状态

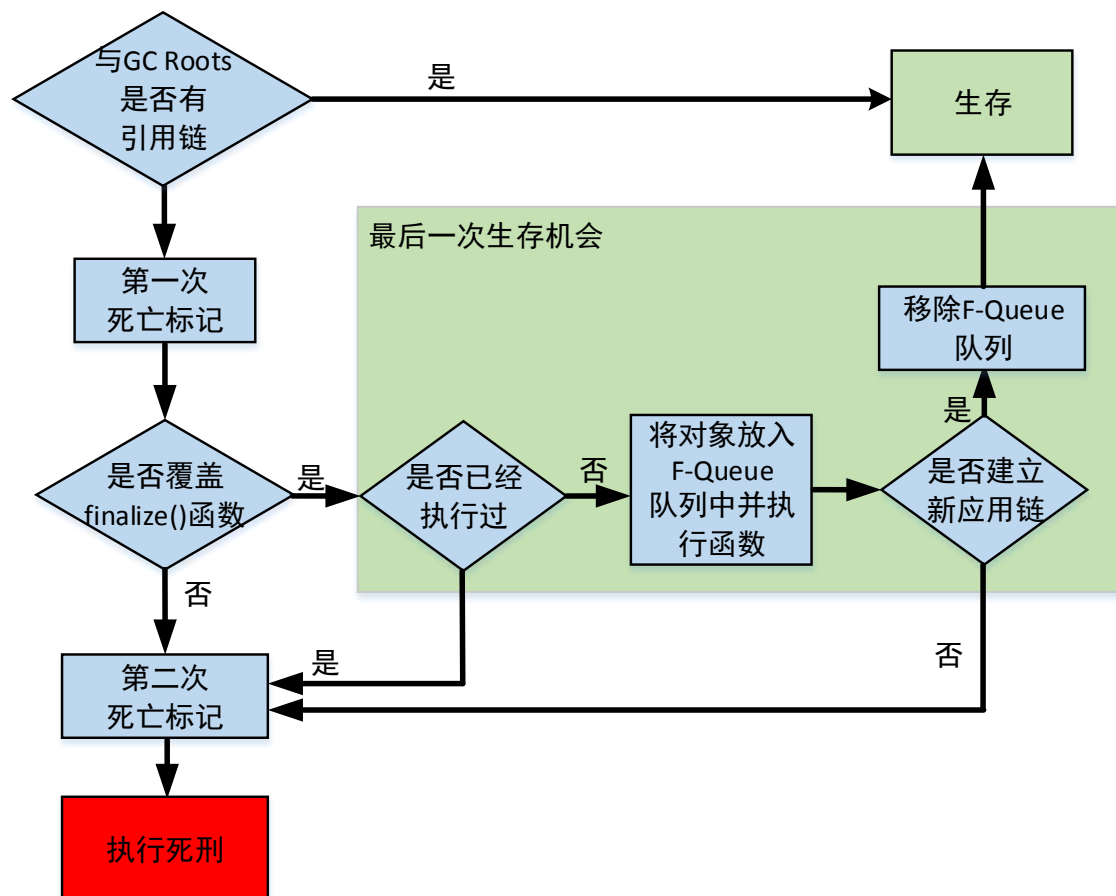
越来越弱

- 强引用：普通的引用。
- 软引用：若内存足够，则不会回收；若内存不够，则回收。
- 弱引用：只能生存到下一次垃圾收集发生之前。
- 虚引用：仅在被回收时收到系统通知。

垃圾收集过程 对象已死吗？



- 生存还是死亡？
 - 不可达的对象，也并非“非死不可”，暂处缓刑。要真正宣告一个对象死亡，至少要经历两次标记过程：



- 标记-清除算法

- 效率低

- 空间问题，产生大量不连续内存碎片，无法给大对象分配。

内存整理前

内存整理后

可用内存

可回收内存

存活对象

- 复制算法

- 将内存分为两块，每次只使用其中的一块。
- 实现简单，运行高效，但是将内存缩小为原来的一半。

内存整理前

内存整理后

可用内存	可回收内存	存活对象	保留内存
------	-------	------	------

- 标记-整理算法

- 先标记，然后从内存中0地址开始，复制存活对象；
- 然后直接清理掉端边界以外的所有内存。

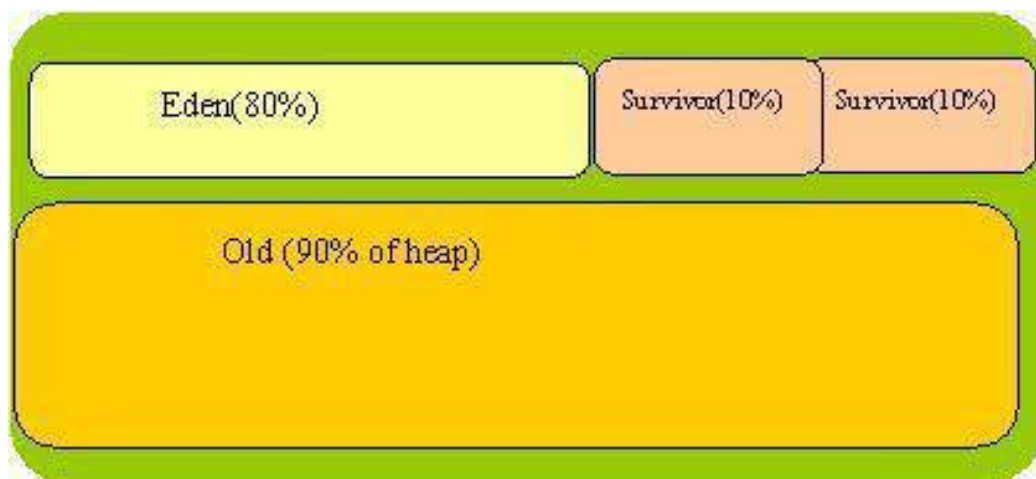
内存整理前

内存整理后

可用内存	可回收内存	存活对象
------	-------	------

- 分代收集算法

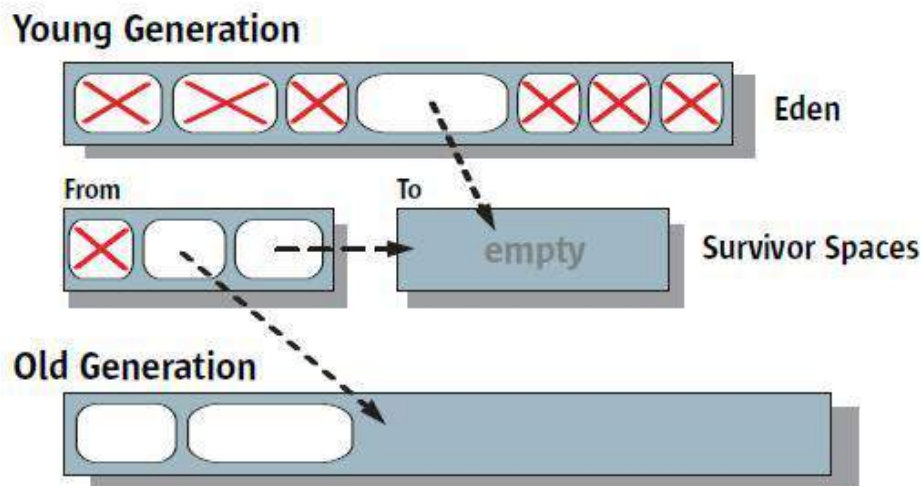
- 当前商业虚拟机的垃圾收集都采用“分代收集”算法，即将堆内存划分为**新生代**和**老年代**；
- 新生代中每次垃圾收集时都会有大批对象死去，少量存活，适合用**复制算法**，只需付出少量存活对象的复制成本。
- 老年代中对象存活率高，没有额外空间进行分配担保，则采用“**标记-清除**”或“**标记-整理**”算法进行回收。



- 分代收集算法

- 新生代分为Eden区和Survivor区；
- Survivor区分为From区和To区；这两个区至少有一个为空的，每次GC时，Eden区和非空的Survivor区中存活的对象则进入空的Survivor区；若空间不够，则提前进入老年代。
- 新对象优先在Eden区分配，年龄为0，每经过一次GC年龄加1，
对象直接进

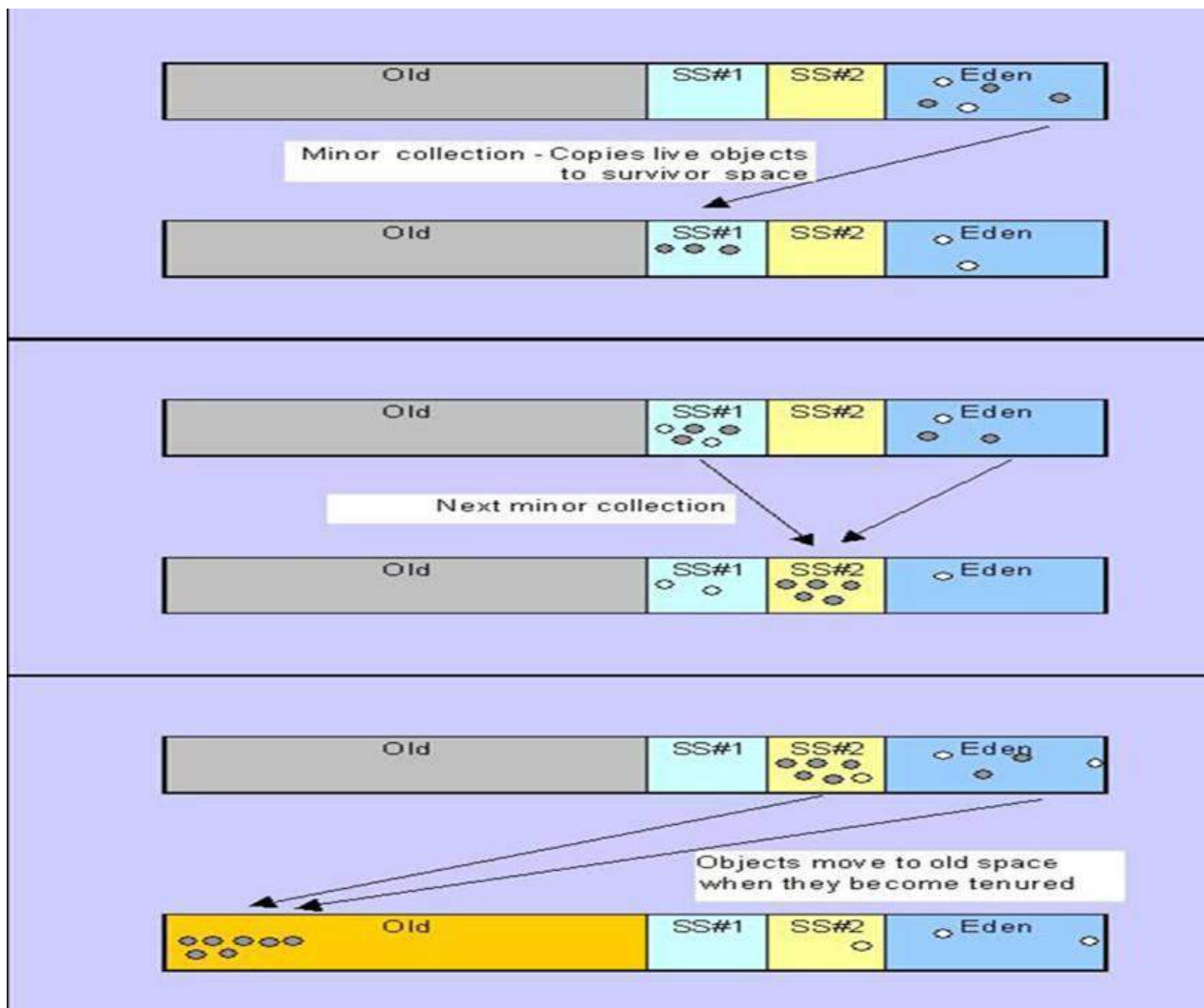
入老年代



垃圾收集过程 垃圾收集算法



- 分代收集算法



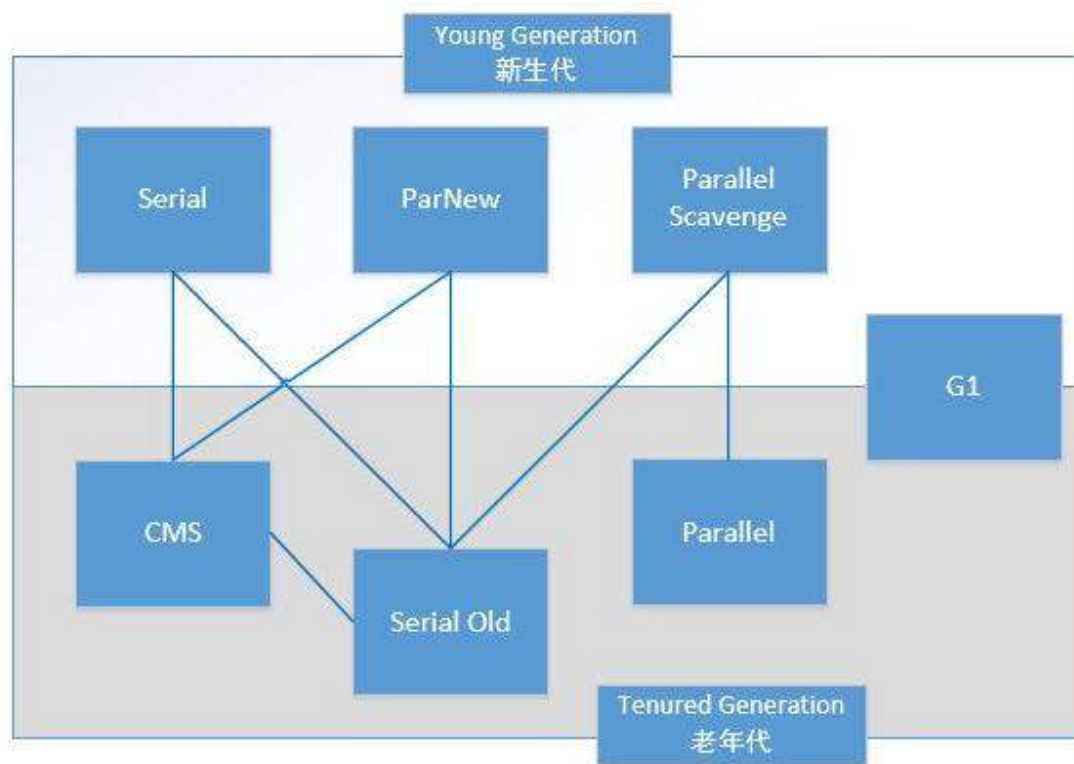
- 分代收集算法

- 新生代发生的GC称为Minor GC，当出现以下情况时会触发：
 - Eden区没有足够空间进行分配时；
- 老年代发生的GC称为Full GC，当出现以下情况时会触发：
 - 调用System.gc时；
 - 老年代空间不足；
 - 方法区空间不足；
 - 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；
 - 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。

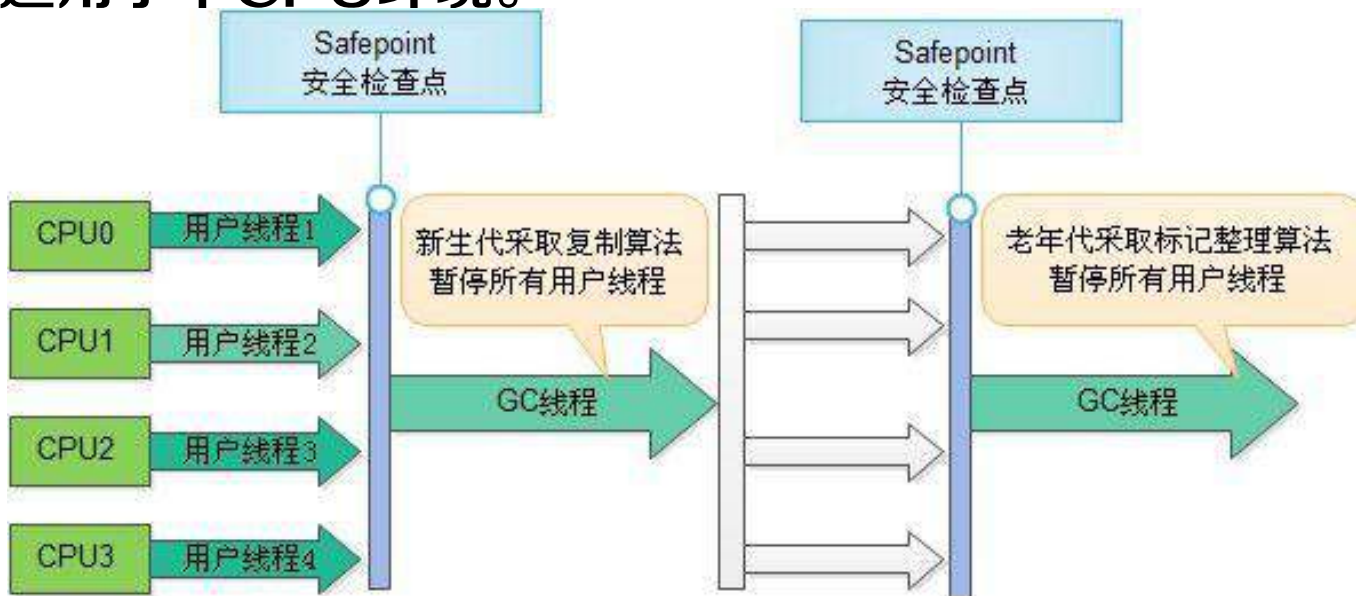


垃圾收集器

- 如果说收集算法是内存回收的方法论，那么垃圾收集器就是内存回收的具体实现。
- Java虚拟机规范中对垃圾收集器应该如何实现没有任何规定，因此不同厂商、版本的虚拟机提供的垃圾收集器都不同。



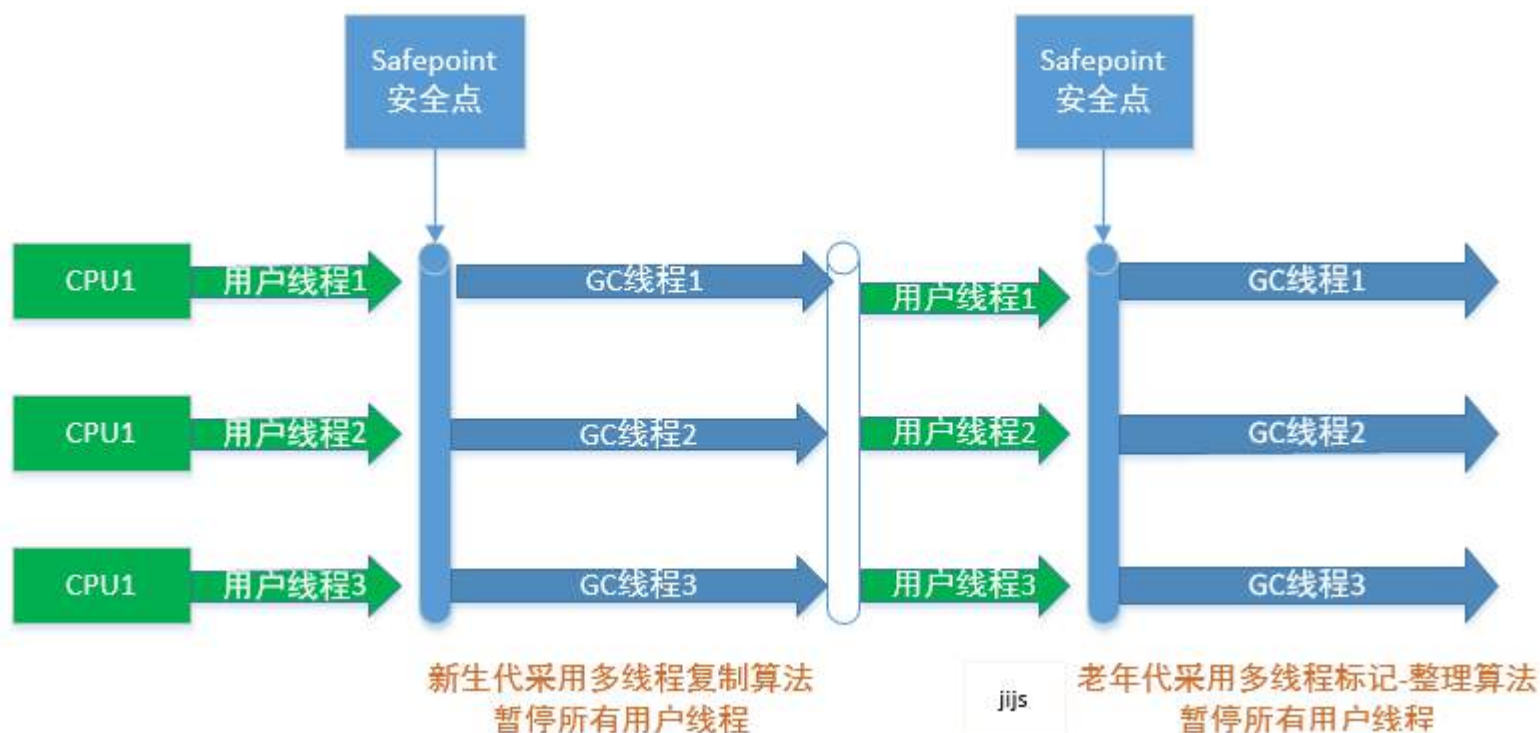
- Serial/Serial Old收集器（JDK1.3之前）
 - 单线程收集器；
 - 新生代使用复制算法，老年代使用标记-整理算法；
 - 进行垃圾收集时，必须暂停所有的线程(Stop The World)。
 - 适用于单CPU环境。



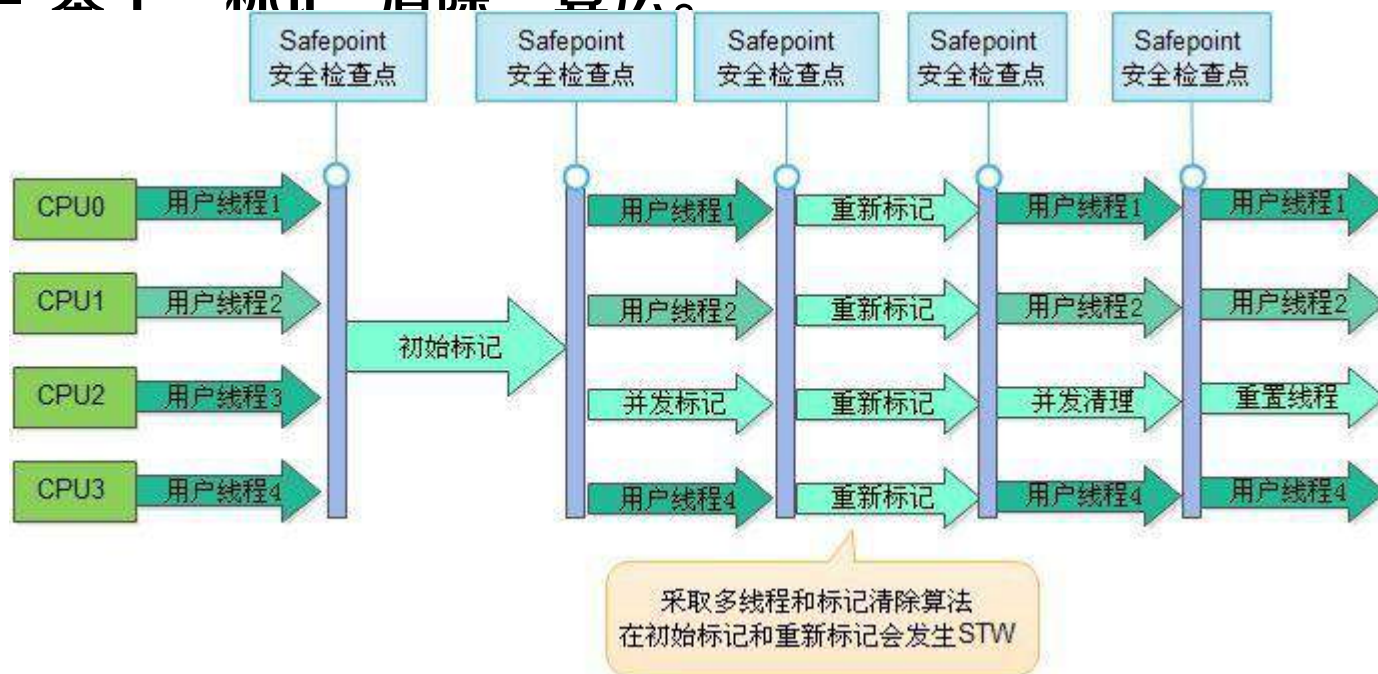
- Parnew/Parallel scavenge（并行回收）收集器
 - Serial收集器的多线程版本；
 - 其中Parallel scavenge收集器可通过设置相关参数来达到目标吞吐量， $\text{吞吐量} = \text{程序运行时间} / (\text{程序运行时间} + \text{垃圾收集时间})$



- Parallel old收集器
 - Parallel Scavenge收集器的老年代版本；
 - 使用多线程和“标记-整理”算法。



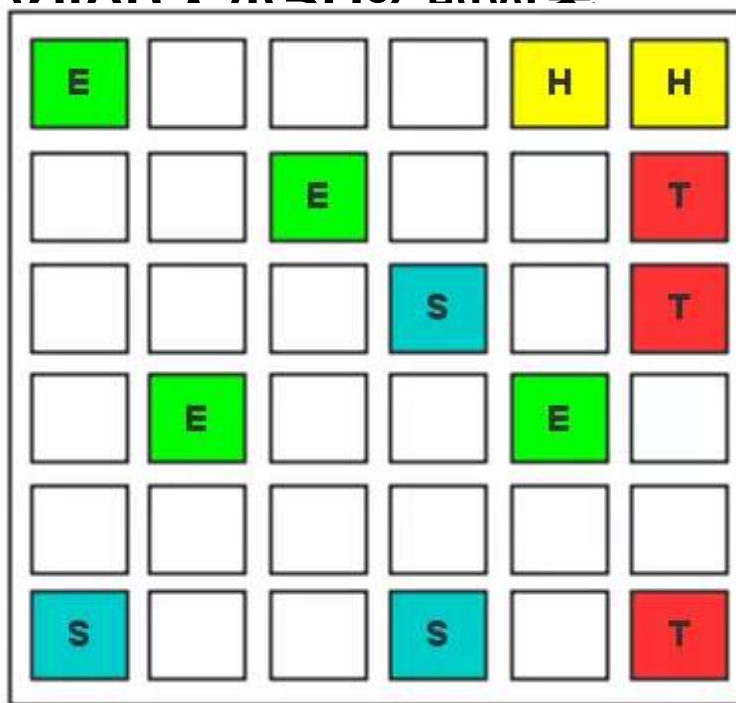
- CMS(Concurrent Mark Sweep并发标记清除)收集器(JDK1.5)
 - 几乎真正的实现并发；
 - 以最短回收停顿时间为目标；
 - 基于“标记-清除”算法。



- CMS(Concurrent Mark Sweep 并发标记清除)收集器
 - **初始标记**: 只标记GC Roots直接关联的对象, **需Stop The World**, 耗时很短;
 - **并发标记**: 进行GC Roots根搜索算法, 判定对象是否存活, 耗时稍长;
 - **重新标记**: 修正并发标记期间, 因用户程序继续运行导致标记产生变动的那一部分对象的标记记录; **需Stop The World**, 耗时比初始标记稍长, 但比并发标记短;
 - **并发清除**: 对标记死亡的对象进行清除。
 - **优点**: 并发收集, 低停顿;
 - **缺点**: 对CPU资源敏感, 无法处理浮动垃圾, 产生大量碎片。

- G1(Garbage First)收集器
 - 将整个Java堆划分为多个大小相等的**独立区域** (Region)，虽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，他们都是一部分Region (不需要连续) 的集合。
 - 有计划的避免在Java堆中进行全区域的垃圾收集。G1跟踪各个Region的**垃圾堆积价值大小**，在后台维护一个**优先列表**，每次根据允许的收集时间，优先回收价值最大的Region，保证最高的收集效率。
 - G1是目前收集器技术发展的最前沿成果之一，但这种“化整为零”的逻辑理解起来容易，实现起来很难，2014年耗费10年时间开发出G1商用版。

- G1(Garbage First)收集器
 - 图中E表示Eden区，S表示Survivor区，T表示Tenured区；
 - 添加了H表示Humongous区，主要用于存储大对象-即大小超过了region大小50%的对象



• 收集器应用汇总

序号	收集器	收集范围	算法	执行类型
1	Serial	新生代	复制	单线程
2	ParNew	新生代	复制	多线程并行
3	Parallel	新生代	复制	多线程并行
4	Serial Old	老年代	标记整理	单线程
5	CMS	老年代	标记清除	多线程并发
6	Parallel Old	老年代	标记整理	多线程
7	G1	全部	复制算法，标记-整理	多线程



[1] 深入理解Java虚拟机 JVM高级特性与最佳实践 周志明

[2] <http://blog.csdn.net/shakespeare001/article/details/51749788>

[3] <http://www.th7.cn/Program/java/201705/1167843.shtml>

上善若水。水善利万物
而不争，处众人之所恶，
故几於道。居善地，心
善渊与善仁，言善信，
正善治，事善能，动善
时。夫唯不争，故无尤。

谢谢！

