咕泡学院
gupaoedu.com

专业互联网
IT 教育服务平台

[VIP课]

JVM入门与实战

DO ONE THING AT A TIME AND DO WELL.

- 大白老师QQ号： 1828627710

自我介绍

咕泡学院- 大白老师

前大众点评架构师

十余年Java经验，曾任职于1号店、大众点评、同程旅游、阿里系公司，担任过技术总监、首席架构师、team leader、系统架构师。有着多年的前后台大型分布式项目架构经验，在处理高并发、性能调优上有独到的方法论。精通Java、J2EE架构、Redis、MongoDB、Netty，消息组件如Kafka、RocketMQ。

课程目标

- 了解JVM内存模型以及每个分区详解
- 熟悉运行时数据区，特别是堆内存结构和特点
- 熟悉GC三种收集方法的原理和特点
- 熟练使用GC调优工具，快速诊断线上问题
- 生产环境CPU负载升高怎么处理
- 生产环境给应用分配多少线程合适
- JVM字节码是什么东西

1

JVM入门与实战

JVM入门与实战

1 JVM体系结构概述

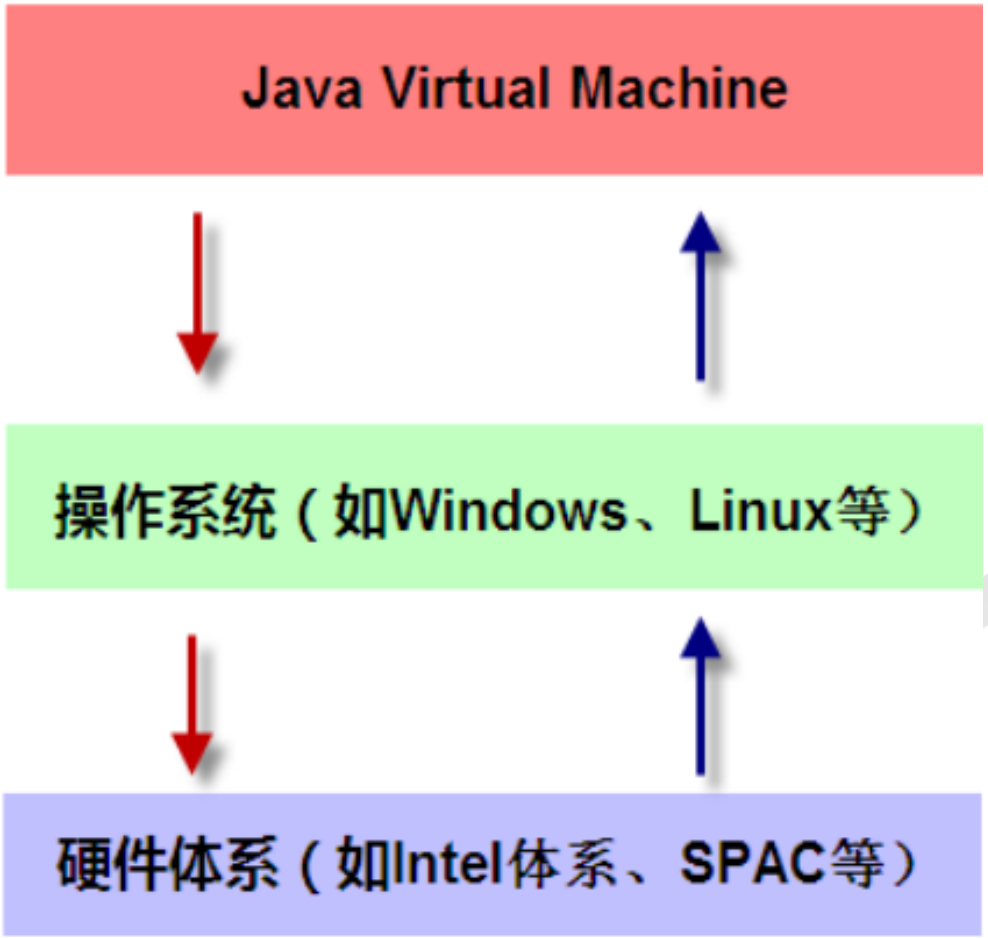
2 堆体系结构概述

3 GC参数调优入门

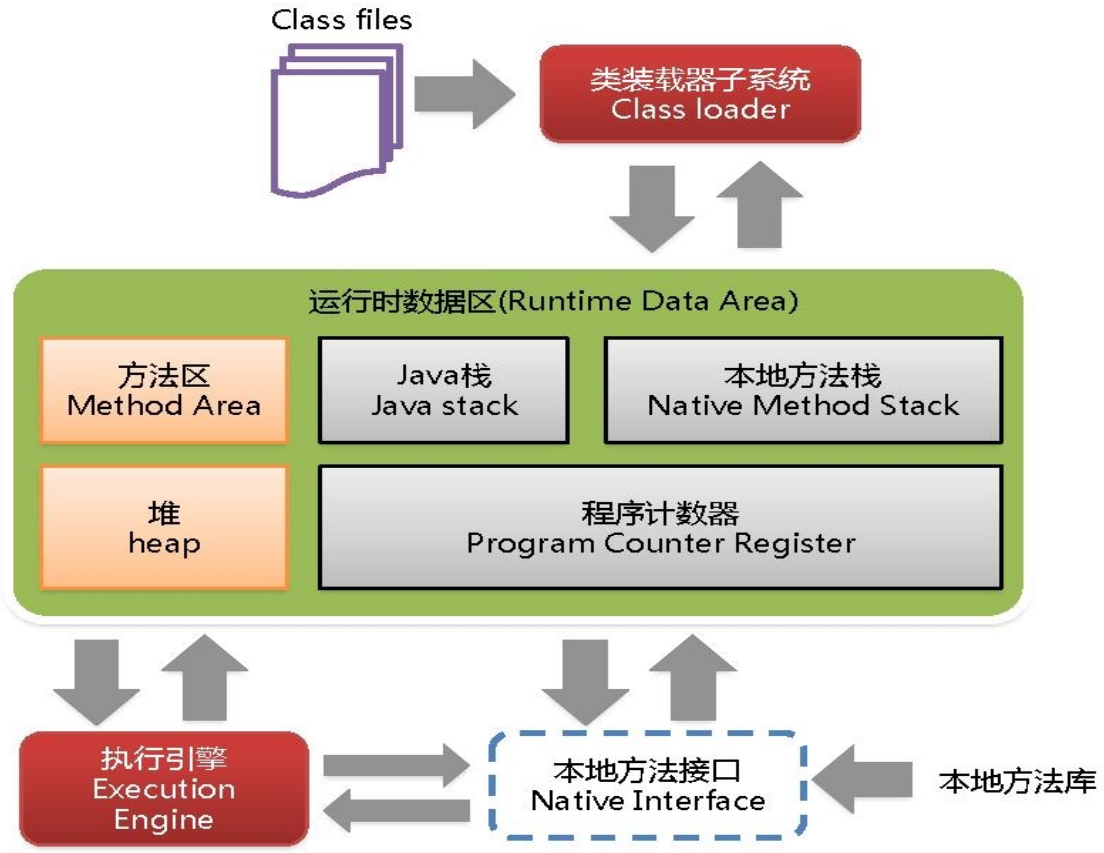
4 总　结

JVM体系结构概述

- JVM位置

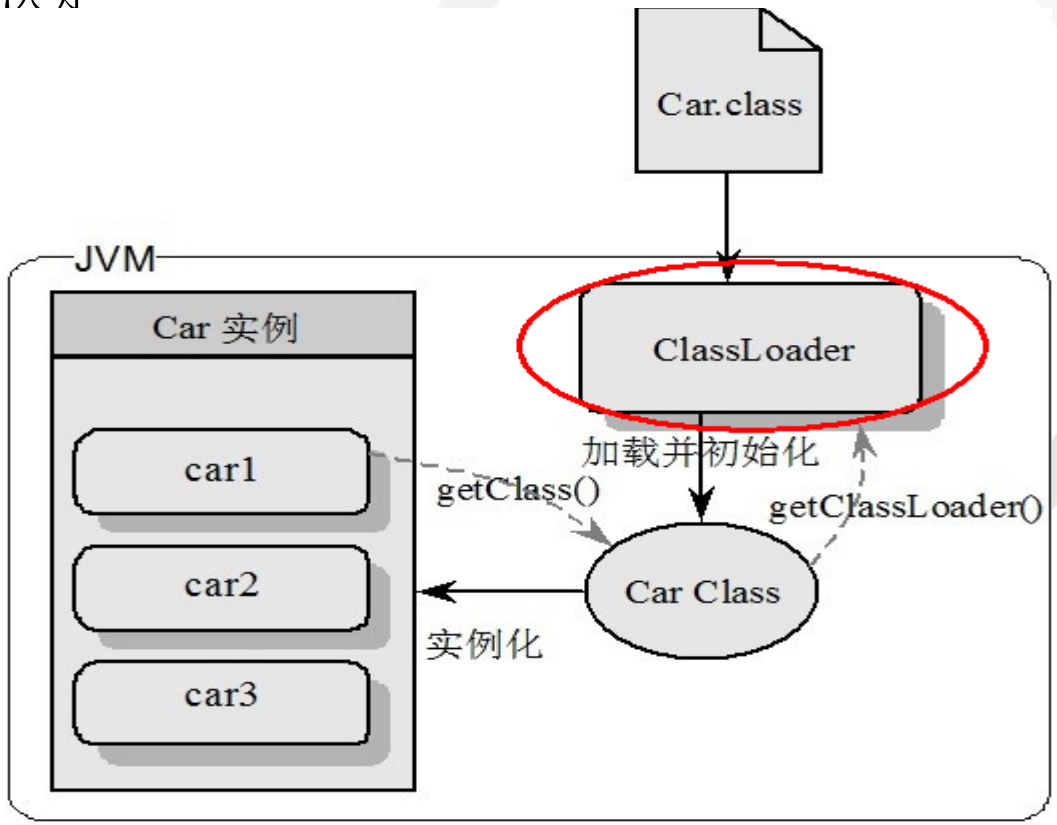


JVM体系结构概览

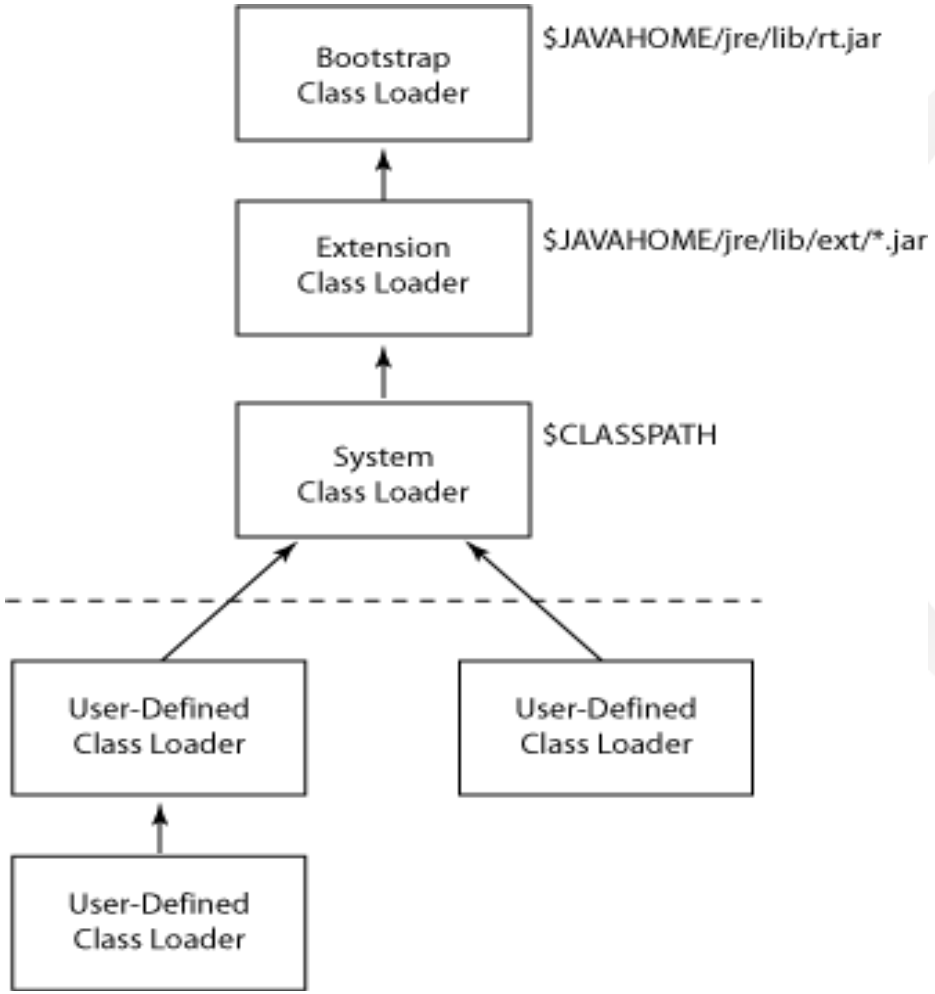


类装载器ClassLoader

负责加载class文件，class文件在文件开头有特定的文件标示，并且ClassLoader只负责class文件的加载，至于它是否可以运行，则由Execution Engine决定



类装载器ClassLoader2



- 虚拟机自带的加载器
- 启动类加载器（Bootstrap）C++
- 扩展类加载器（Extension）Java
- 应用程序类加载器（AppClassLoader）Java

也叫系统类加载器，加载当前应用的classpath的所有类

- 用户自定义加载器

Java.lang.ClassLoader的子类，用户可以定制类的加载方式

Execution Engine

执行引擎负责解释命令，提交操作系统执行

PC寄存器

- 程序计数器是一块较小的内存空间，是当前线程所执行的字节码的行号指示器
- 程序计算器处于线程独占区
- 如果线程执行的是java方法，记录的是正在执行的虚拟机字节码指令的地址，如果是native方法，这个计数器值为undefined

栈区

***Stack** 栈是什么*

栈也叫栈内存，主管Java程序的运行，是在线程创建时创建，它的生命期是跟随线程的生命期，线程结束栈内存也就释放，对于栈来说不存在垃圾回收问题，只要线程一结束该栈就Over，生命周期和线程一致，是线程私有的。8种基本类型的变量+对象的引用变量+实例方法都是在函数的栈内存中分配。

栈存储什么?

- 局部变量表:输入参数和输出参数以及方法内的变量类型；局部变量表在编译期间完成分配，当进入一个方法时，这个方法在帧中分配多少内存是固定的
- 栈操作（Operand Stack）:记录出栈、入栈的操作；
- 动态链接
- 方法出口

栈溢出

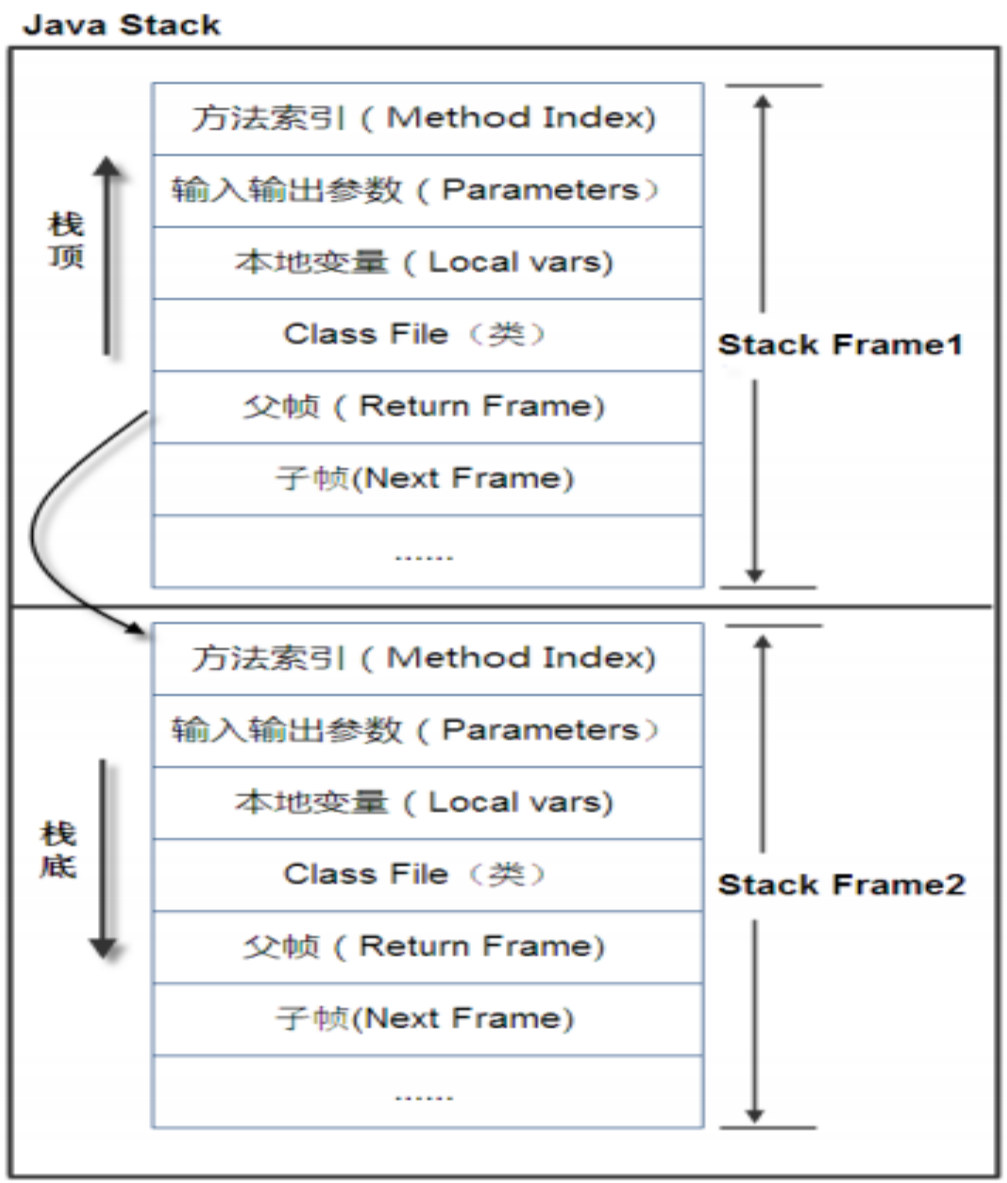
StackOverflowError,OutOfMemory

Native Interface本地接口

本地接口的作用是融合不同的编程语言为 Java 所用

Native Method Stack

它的具体做法是Native Method Stack中登记native方法，在Execution Engine 执行时加载本地方法库。



图示在一个栈中有两个栈帧：

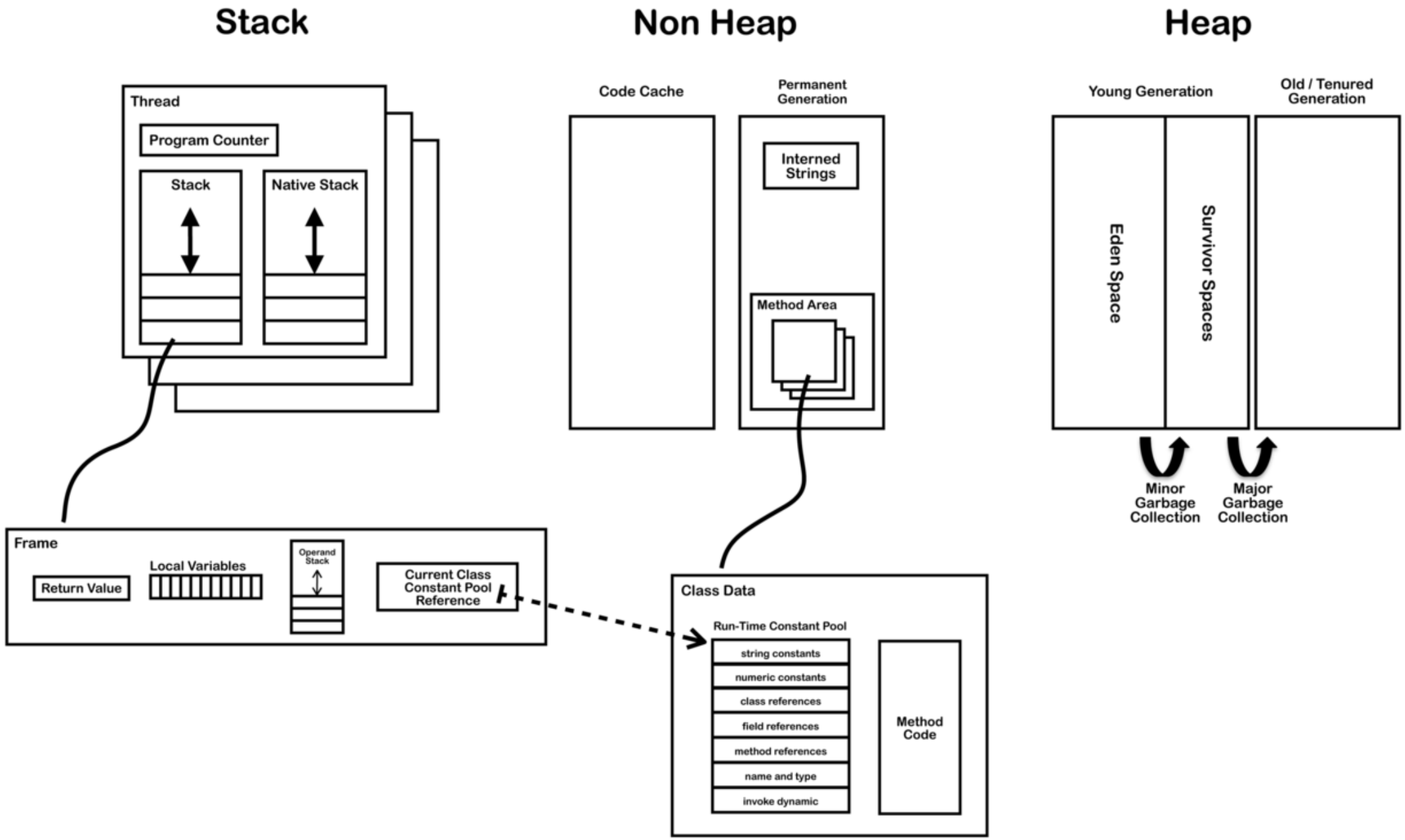
栈帧 2是最先被调用的方法，先入栈，

然后方法 2 又调用了方法1，栈帧 1处于栈顶的位置，

栈帧 2 处于栈底，执行完毕后，依次弹出栈帧 1和栈帧 2，

线程结束，栈释放。

每执行一个方法都会产生一个栈帧，保存到栈(后进先出)的顶部，顶部栈就是当前的方法，该方法执行完毕 后会自动将此栈帧出栈。


Stack
Non Heap
Heap
Thread
Program Counter
Stack
Native Stack
Code Cache
Permanent Generation
Interned Strings
Method Area
Young Generation
Old / Tenured Generation
Eden Space
Survivor Spaces
Minor Garbage Collection
Major Garbage Collection
Frame
Return Value
Local Variables
Operand Stack
Current Class Constant Pool Reference
Class Data
Run-Time Constant Pool
string constants
numeric constants
class references
field references
method references
name and type
invoke dynamic
Method Code

方法区

Method Area

方法区是被所有线程共享，所有字段和方法字节码，以及一些特殊方法如构造函数，接口代码也在此定义。简单说，所有定义的方法的信息都保存在该区域，此区属于共享区间。

- 类信息
  类的版本
  字段
  方法
  接口
- 静态变量
- 常量
- 类信息(构造方法/接口定义)
- 运行时常量池

方法区与永久代

方法区

永久存储区是一个常驻内存区域，用于存放JDK自身所携带的 Class,Interface 的元数据，也就是说它存储的是运行环境必须的类信息，被装载进此区域的数据是不会被垃圾回收器回收掉的，关闭 JVM 才会释放此区域所占用的内存。

如果出现java.lang.OutOfMemoryError: PermGen space，说明是Java虚拟机对永久代Perm内存设置不够。一般出现这种情况，都是程序启动需要加载大量的第三方jar包。例如：在一个Tomcat下部署了太多的应用。或者大量动态反射生成的类不断被加载，最终导致Perm区被占满。

Jdk1.6及之前： 有永久代, 常量池1.6在方法区
Jdk1.7： 有永久代，但已经逐步“去永久代”，常量池1.7在堆
Jdk1.8及之后： 无永久代，常量池1.8在元空间

方法区-运行时常量池

它是方法区的一部分，用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后存放到常量池中

方法区（Method Area），是各个线程共享的内存区域，它用于存储虚拟机加载的：类信息+普通常量+静态常量+编译器编译后的代码等等，虽然JVM规范将方法区描述为堆的一个逻辑部分，但它却还有一个别名叫做Non-Heap(非堆)，目的就是要和堆分开。

对于HotSpot虚拟机，很多开发者习惯将方法区称之为“永久代(Parmanent Gen)”，但严格本质上说两者不同，或者说使用永久代来实现方法区而已，永久代是方法区(相当于是一个接口interface)的一个实现，jdk1.7的版本中，已经将原本放在永久代的字符串常量池移走。

常量池（Constant Pool）是方法区的一部分，Class文件除了有类的版本、字段、方法、接口等描述信息外，还有一项信息就是常量池，这部分内容将在类加载后进入方法区的运行时常量池中存放。

Heap 堆

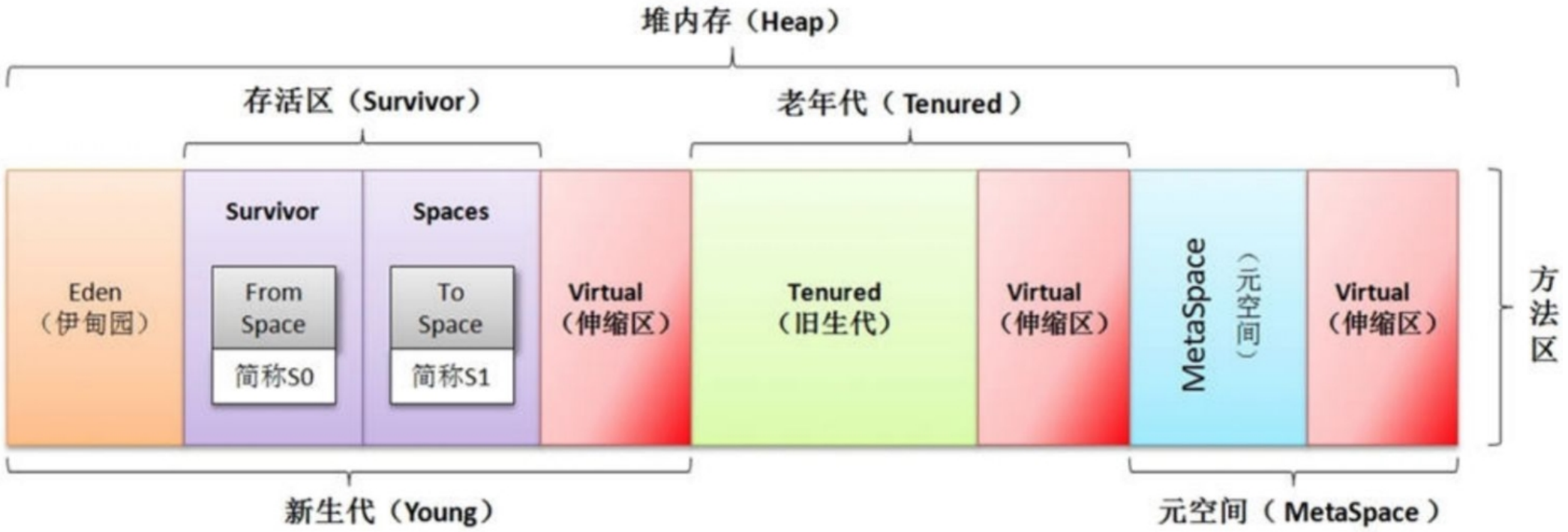
一个JVM实例只存在一个堆内存，堆内存的大小是可以调节的。类加载器读取了类文件后，需要把类、方法、常变量放到堆内存中，保存所有引用类型的真实信息，以方便执行器执行，堆内存分为三部分：

- Young Generation Space 新生区 Young/New
- Tenure generation space 养老区 Old/ Tenure
- Permanent Space 永久区 Perm

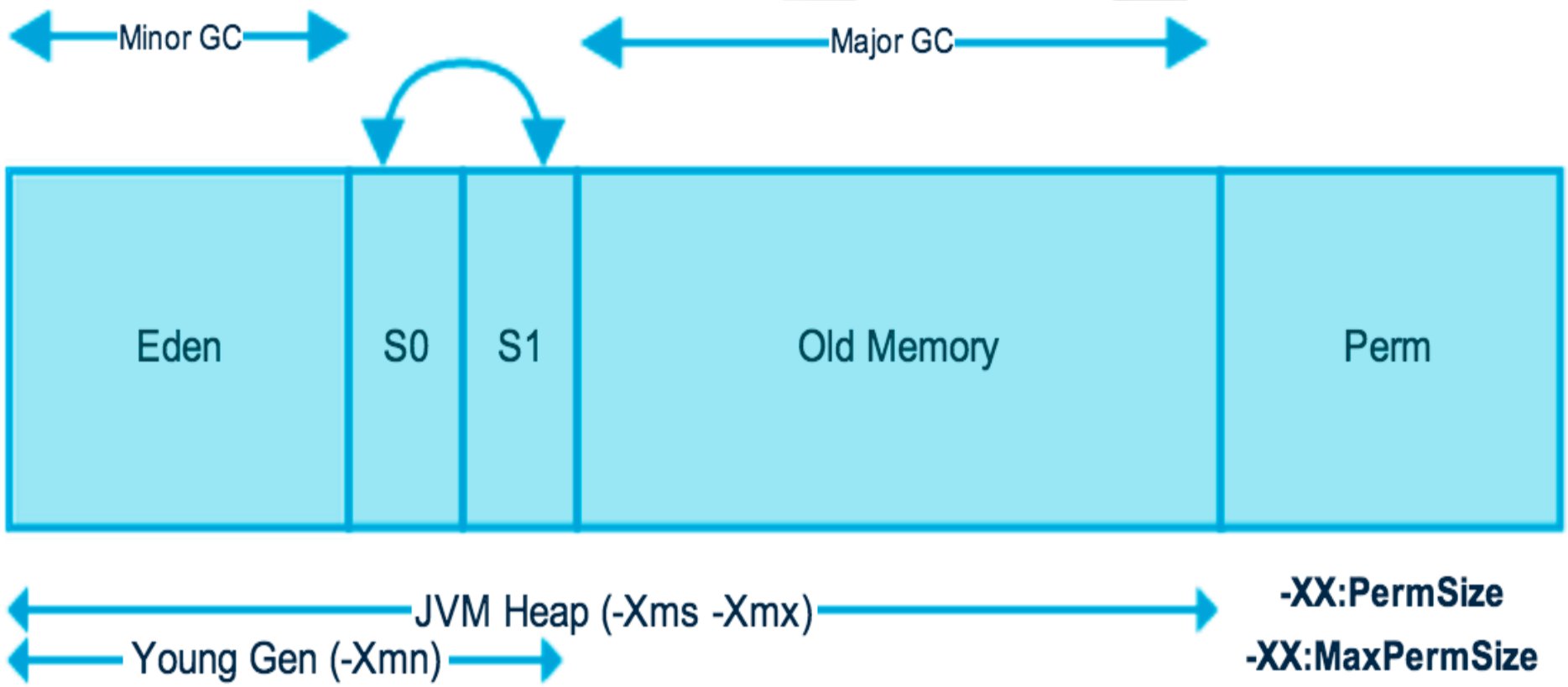
Heap堆(Java8)

一个JVM实例只存在一个堆内存，堆内存的大小是可以调节的。类加载器读取了类文件后，需要把类、方法、常变量放到堆内存中，保存所有引用类型的真实信息，以方便执行器执行。

堆内存逻辑上分为三部分：新生+养老+方法区

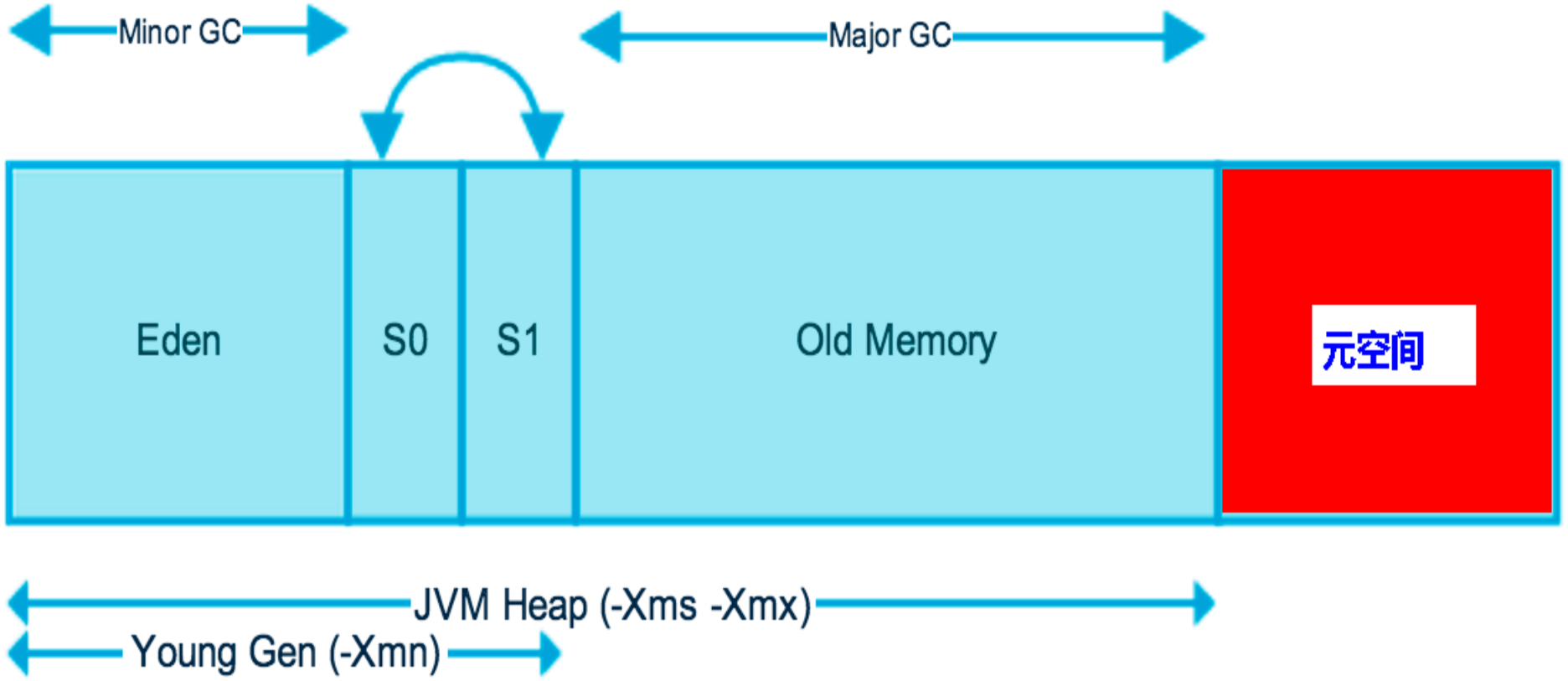


Java7


Minor GC
Major GC
Eden
S0
S1
Old Memory
Perm
JVM Heap (-Xms -Xmx)
Young Gen (-Xmn)
-XX:PermSize
-XX:MaxPermSize

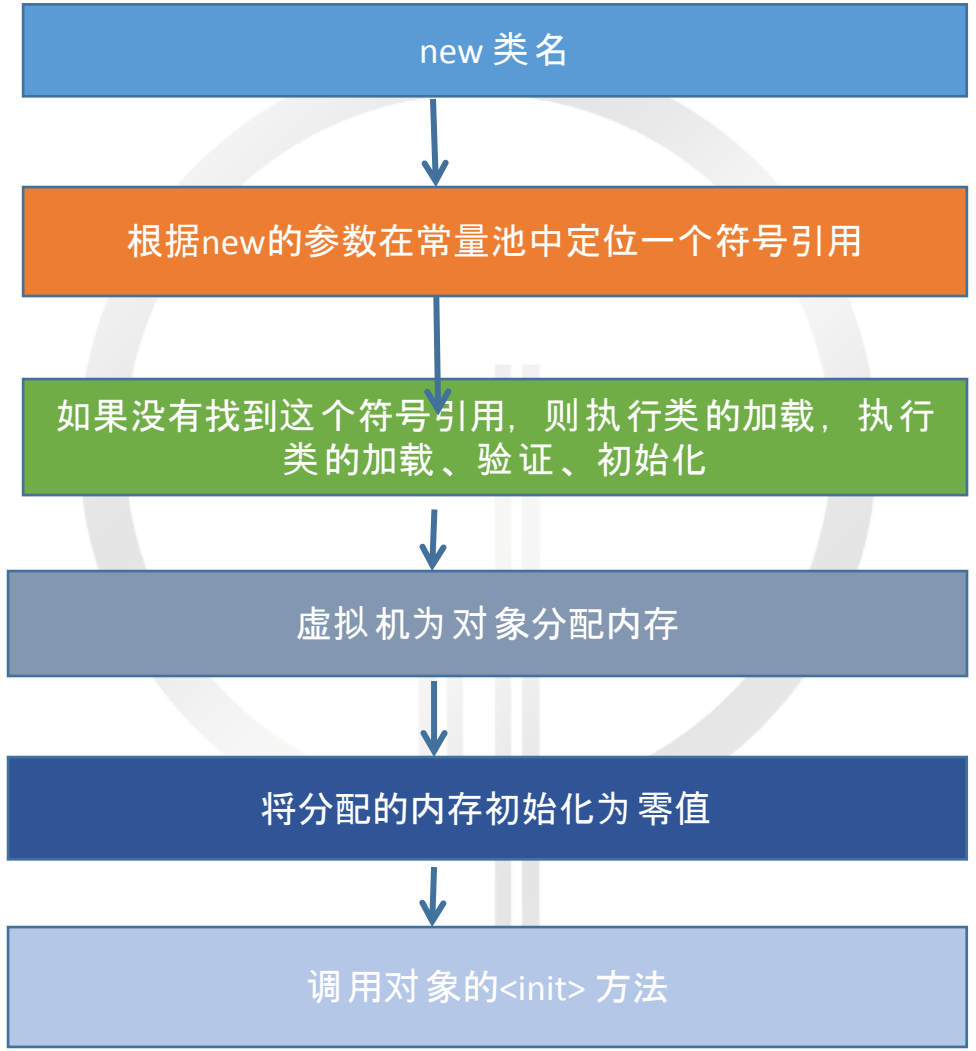
Java8

JDK 1.8之后将最初的永久代取消了，由元空间取代。



对象创建

- 给对象分配内存
- 线程安全性问题
- 初始化对象
- 执行构造方法

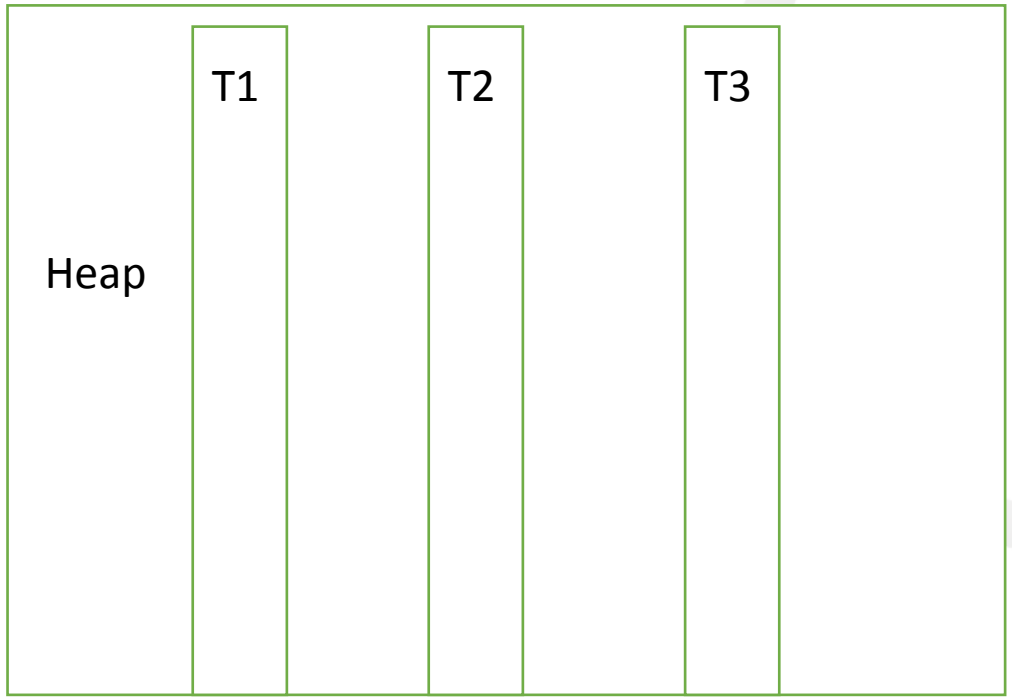


给对象分配内存

- 指针碰撞
- 空间列表

线程安全性问题

- 线程同步
- 本地线程分配缓冲(TLAB)



对象的结构

- Header(对象头)
  - 自身运行时数据（Mark Word）
    - 哈希值
    - GC分代年龄
  - 锁状态标志
  - 线程持有锁
    - 偏向线程ID
  - 偏向时间戳
  - 类型指针
  - 数组长度（只有数组对象才有）
- InstanceData
  - 相同宽度的数据分配到一起（long,double）
- Padding（对齐填充）
  - 8个字节的整数倍

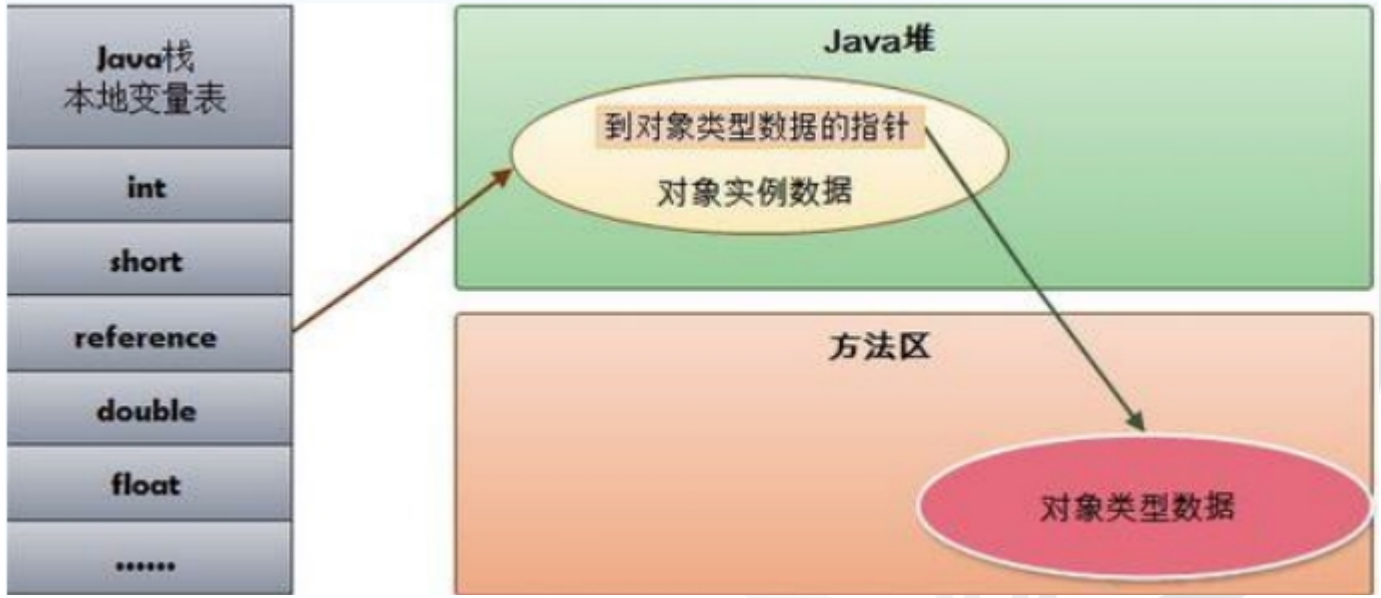
Hotspot虚拟机对象头 Mark Word

锁状态	25bit		4bit	1bit	2bit
	23bit	2bit		是否偏向锁	锁标志位
无锁态	对象的hashCode		分代年龄	0	01
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向互斥量(重量级锁)的指针				10
GC标记	空				11
偏向锁	线程ID	Epoch	分代年龄	1	01

对象的访问定位

- 使用句柄
- 直接指针

栈+堆+方法区的交互关系



HotSpot是使用指针的方式来访问对象：
Java堆中会存放访问类元数据的地址，
reference存储的就直接是对象的地址

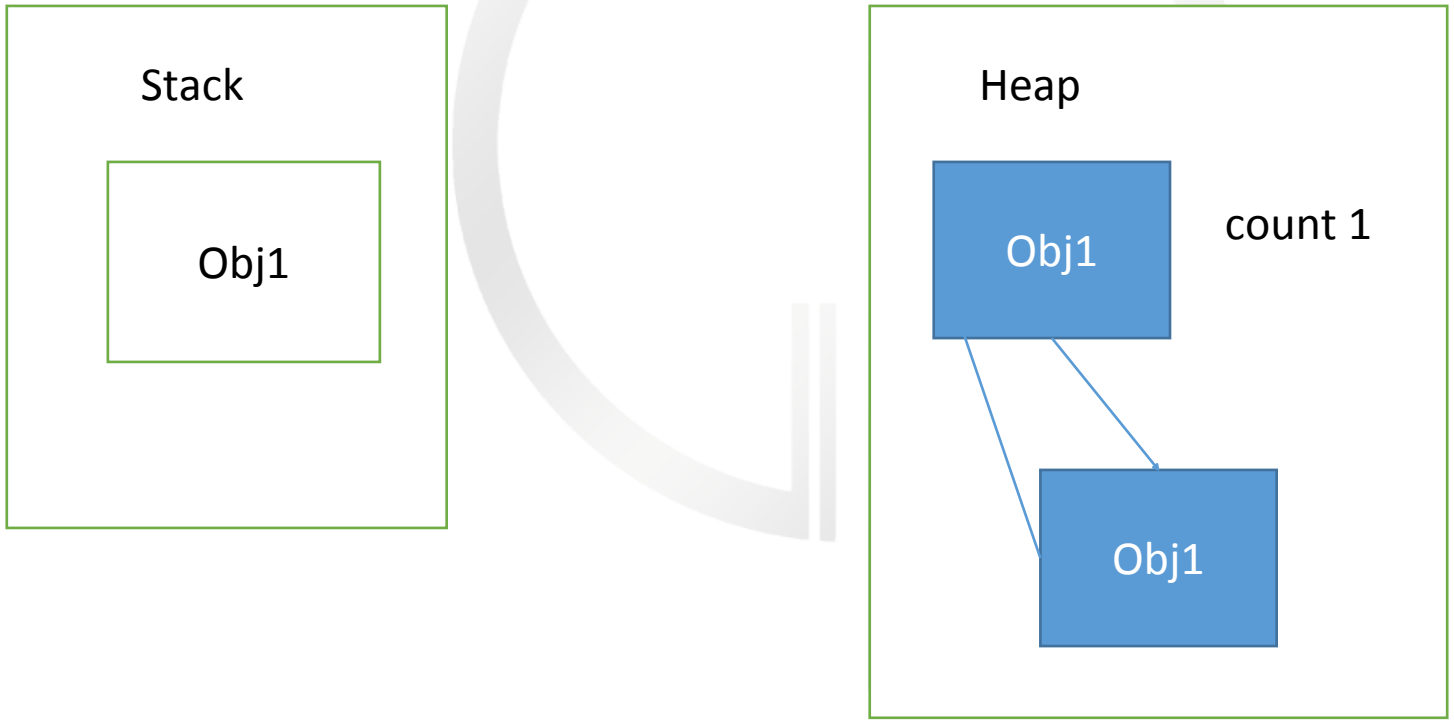
垃圾回收

- 如何判断对象为垃圾对象
  - 引用计数法
  - 可达性分析
- 如何回收
  - 回收策略
    - 标记清除
    - 复制
    - 标记整理
    - 分代算法
  - 垃圾回收器
    - Serial
    - ParNew
    - CMS
    - G1
    - ZGC
- 何时回收

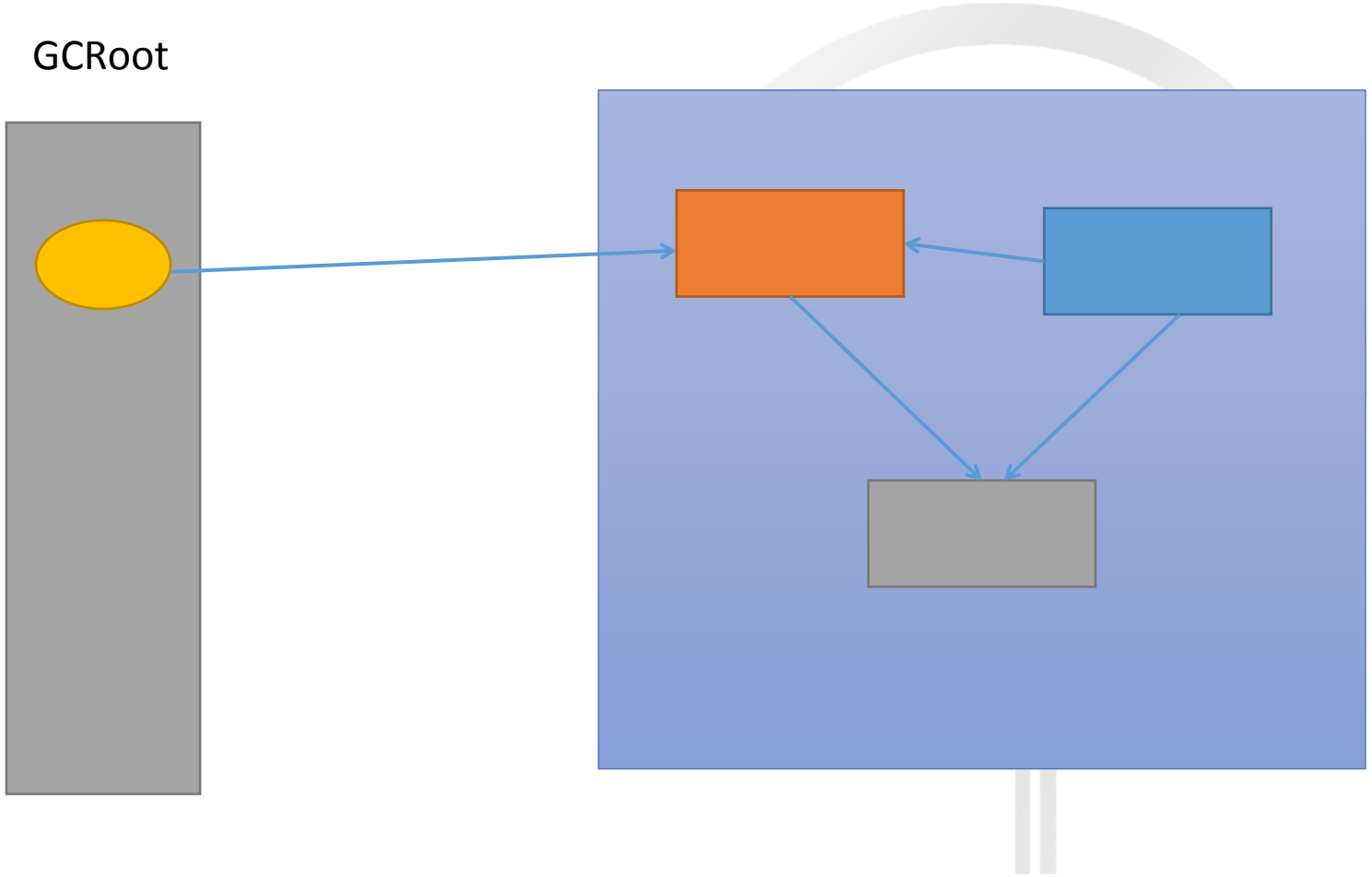
引用计数法

在对象中添加一个引用计数 器，当有地方引用这个对象的时候，计数器+1，当失效的时候，计数器-1

```
-verbose:gc -XX:+PrintGCDetails
```



可达性分析法



可达性分析法

作为GCRoot的对象

- 虚拟机栈(局部变量表中的)
- 方法区的类属性所引用的对象
- 方法区的常量所引用的对象
- 本地方法栈所引用的对象

标记清除算法

分成标记和清除两个阶段
缺点

- 效率问题
- 内存碎片

复制算法

- 堆
  新生代
  Eden
  Survivor
  老年代
  Tenured Gen
- 方法区
- 虚拟机栈
- 本地方法栈
- 程序计数器

标记-整理算法

逃逸分析与栈上分配

内存分配策略

- 优先分配Eden区
- 大对象直接分配到老年代

  -XX:PretenureSizeThreshold
- 长期存活的对象分配老年代

  -XX:MaxTenuringThreshold=15
- 空间分配担保

  -XX:+HandlePromotionFailure

  检查老年代最大可用的连续空间是否大于历次晋升到老年代对象的平均大小。
- 动态对象年龄对象

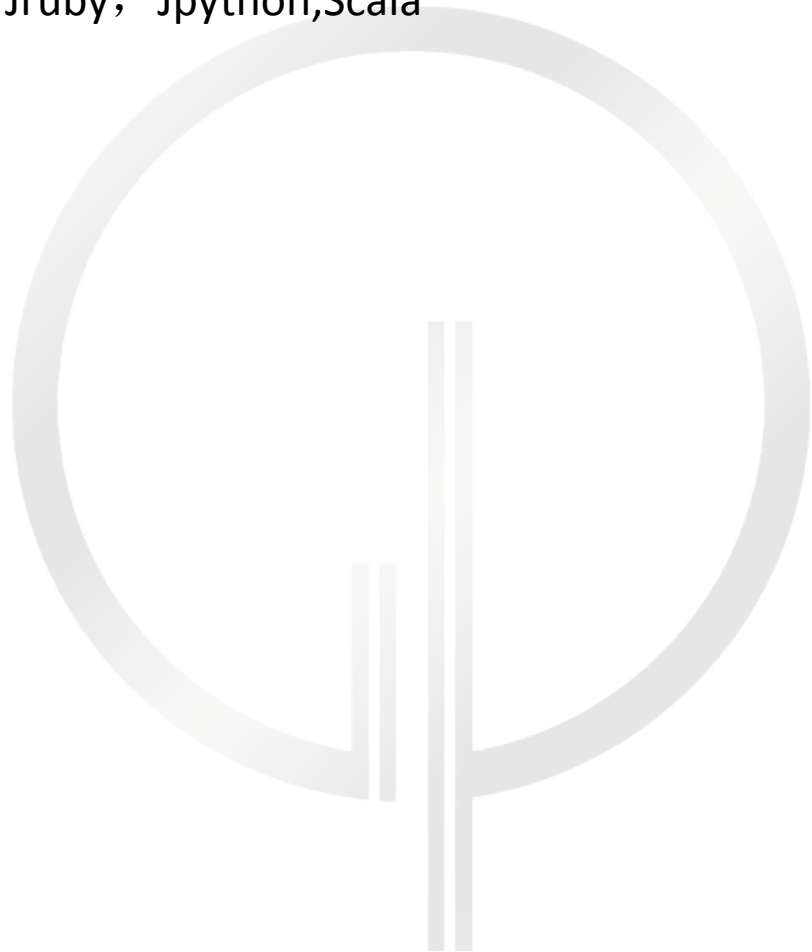
  如果在Survivor空间中相同年龄所有对象大小的总和大于Survivor空间的一半，年龄大于或等于该年龄的对象就可以直接进入老年代

  -XX:TargetSurvivorRatio

性能调优

Class文件设计理念

运行在jvm之上的语言：groovy，Jruby，Jpython,Scala



Class文件简介

java，groovy，jruby，scala

语言

类文件规范

Class文件结构

Class 文件是一组以 8 位字节为基础单位的二进制流，各个数据项目严格按照顺序紧凑地排列在 Class 文件之中，中间没有添加任何分隔符，这使得整个 Class 文件中存储的内容几乎全部是程序运行的必要数据，没有空隙存在。

当遇到需要占用 8 位字节以上空间的数据项时，则会按照高位在前（Big-Endian）的方式分割成若干个 8 位字节进行存储。

Class 文件只有两种数据类型：无符号数和表

链接：
https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-4.html

Class文件结构

魔数
Class文件版本
常量池
访问标志
类索引，父类索引，接口索引集合
字段表集合
方法表集合
属性表集合

Class文件格式

```
ClassFile {
    u4 magic;  // 魔法数字，表明当前文件是.class文件，固定0xCAFEBABE
    u2 minor_version; // 分别为Class文件的副版本和主版本
    u2 major_version;
    u2 constant_pool_count; // 常量池计数
    cp_info constant_pool[constant_pool_count-1];  //
    u2 access_flags; // 类访问标识
    u2 this_class; // 当前类
    u2 super_class; // 父类
    u2 interfaces_count; // 实现的接口数
    u2 interfaces[interfaces_count]; // 实现接口信息
    u2 fields_count; // 字段数量
    field_info fields[fields_count]; // 包含的字段信息
    u2 methods_count; // 方法数量
    method_info methods[methods_count]; // 包含的方法信息
    u2 attributes_count;  // 属性数量
    attribute_info attributes[attributes_count]; // 各种属性
}
```

魔数

版本
JDK1.8 = 52
JDK1.8 = 51

常量池

```
CP_INFO
cp_info {
    u1 tag;
    u1 info[];
}
```

Constant Type	Value
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_String	8
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_NameAndType	12
CONSTANT_Utf8	1
CONSTANT_MethodHandle	15
CONSTANT_MethodType	16
CONSTANT_InvokeDynamic	18

CONSTANT_Methodref_info

用于记录方法的信息

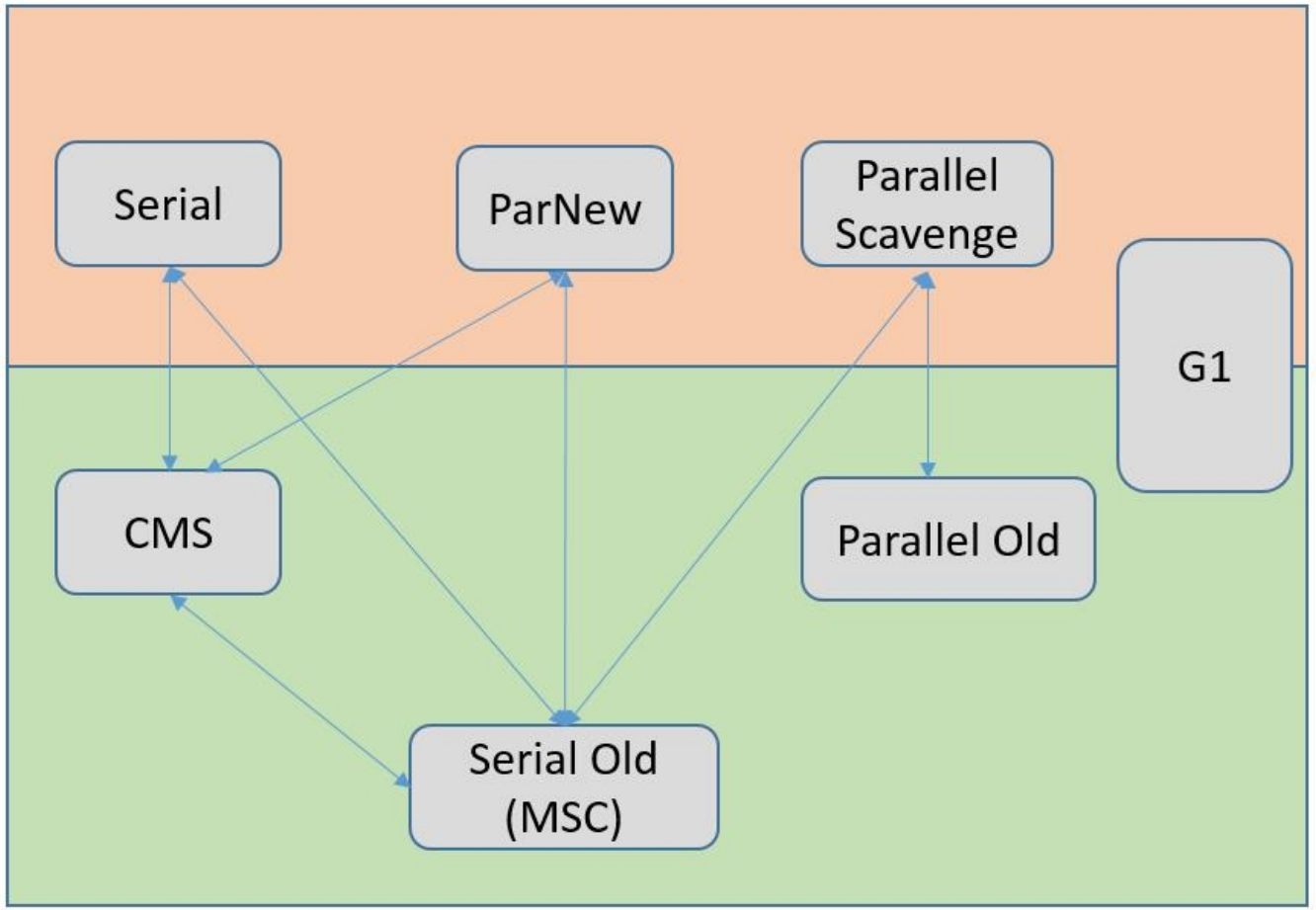
```
CONSTANT_Methodref_info {
    u1 tag;
    u2 class_index;
    u2 name_and_type_index;
}
```


类索引，父类索引

字段表集合

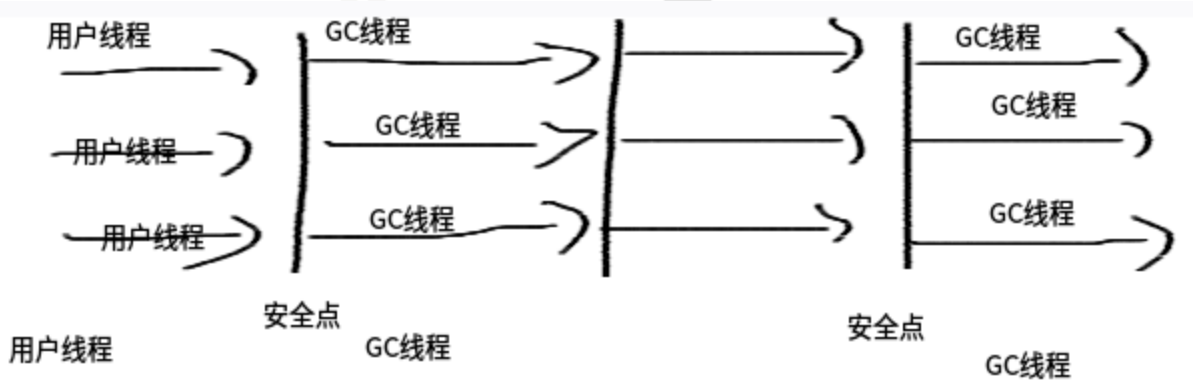
字段表用于描述接口或者类中声明的变量

Hostspot 垃圾收集器



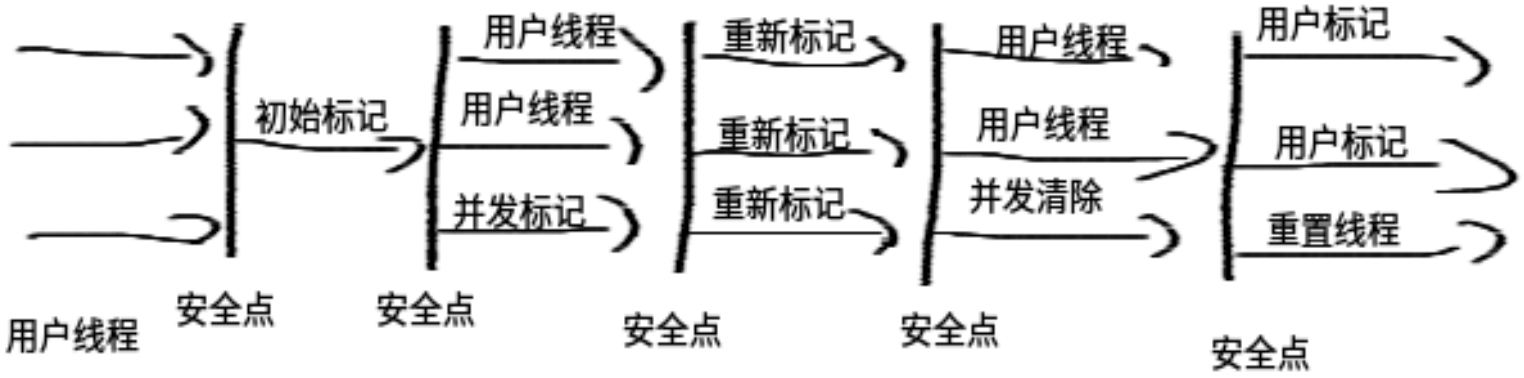
Parallel Scavenge 收集器

- 复制算法(新生代收集器)
- 多线程收集器
- 达到可控制的吞吐量
- 吞吐量：CPU用于运行用户代码时间与CPU消耗总时间的比值
- 吞吐量=执行用户代码时间/(执行用户代码时间 + 垃圾回收使用的时间）
- -XX:MaxGCPauseMillis 垃圾收集停顿时间
- -XX:GCTimeRatio 吞吐量大小 （0,100）



CMS 收集器 (Concurrent Mark Sweep)

- 工作过程
  - 初始标记
  - 并发标记
  - 重新标记
  - 并发清理

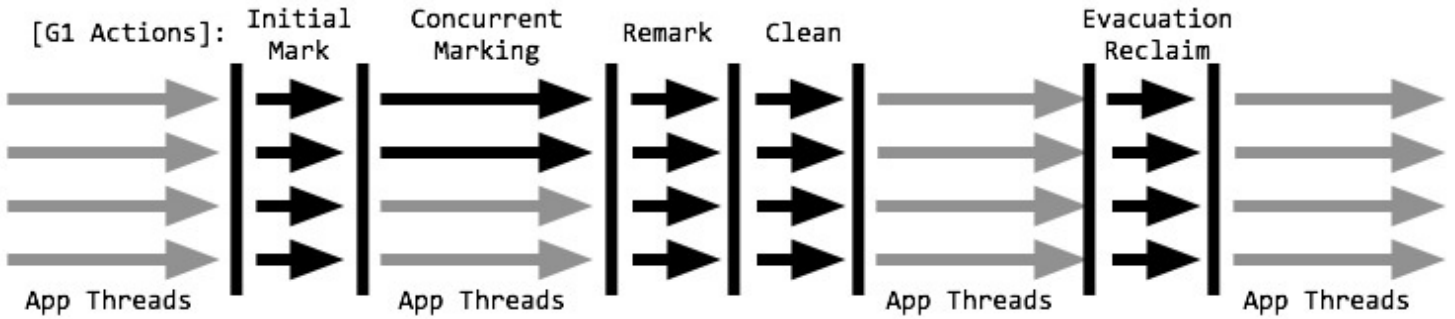


- 优点
  - 并发收集
  - 低停顿

- 缺点
  - 占用CPU资源
  - 无法处理浮动垃圾
  - 出现Concurrent Mode Failure
  - 空间碎片

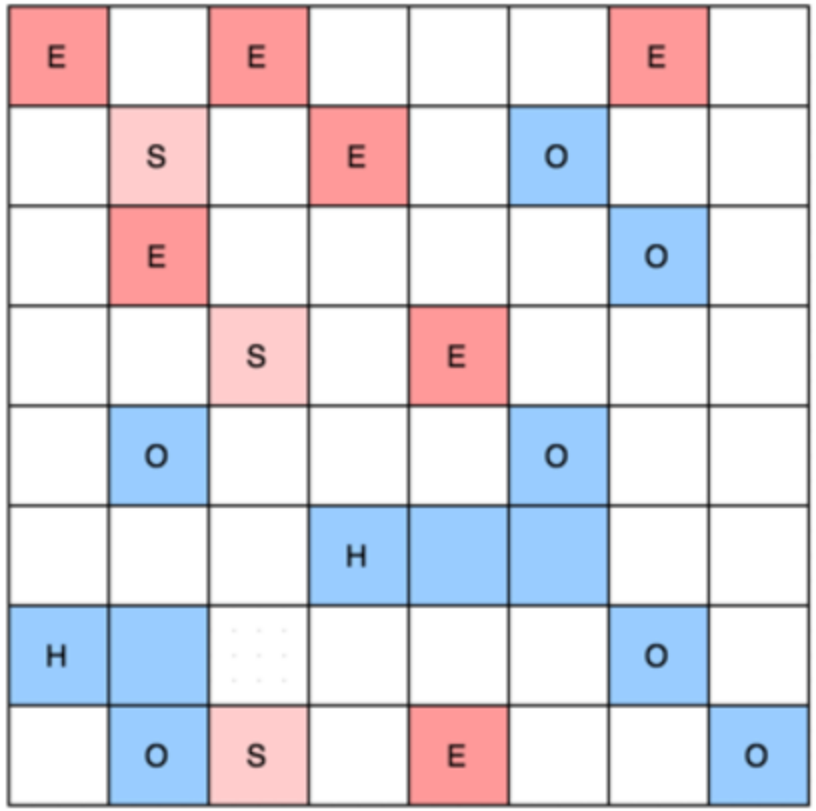
G1 收集器

- 历史
  - 2004年 Sun公司实验室发表了论文
  - JDk7 才使用了G1
- 优势
  - 并行与并发
  - 分代收集
  - 空间整合
  - 可预测的停顿
- 步骤
  - 初始标记
  - 并发标记
  - 最终标记
  - 帅选回收
- 与CMS比较



G1 收集器

- G1的内存模型

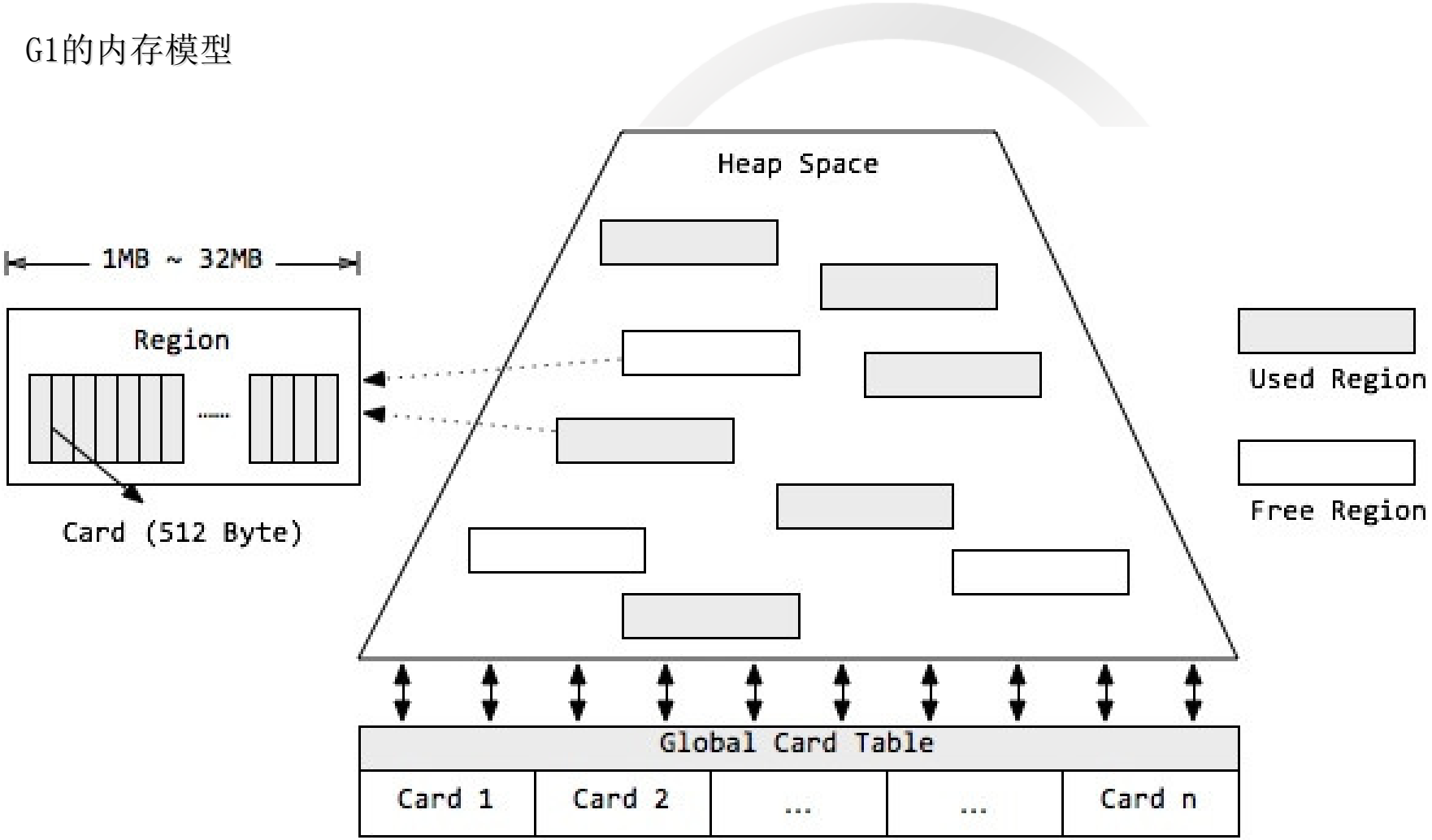


E 新生代空间　S 幸存区空间

O 老年代空间　H

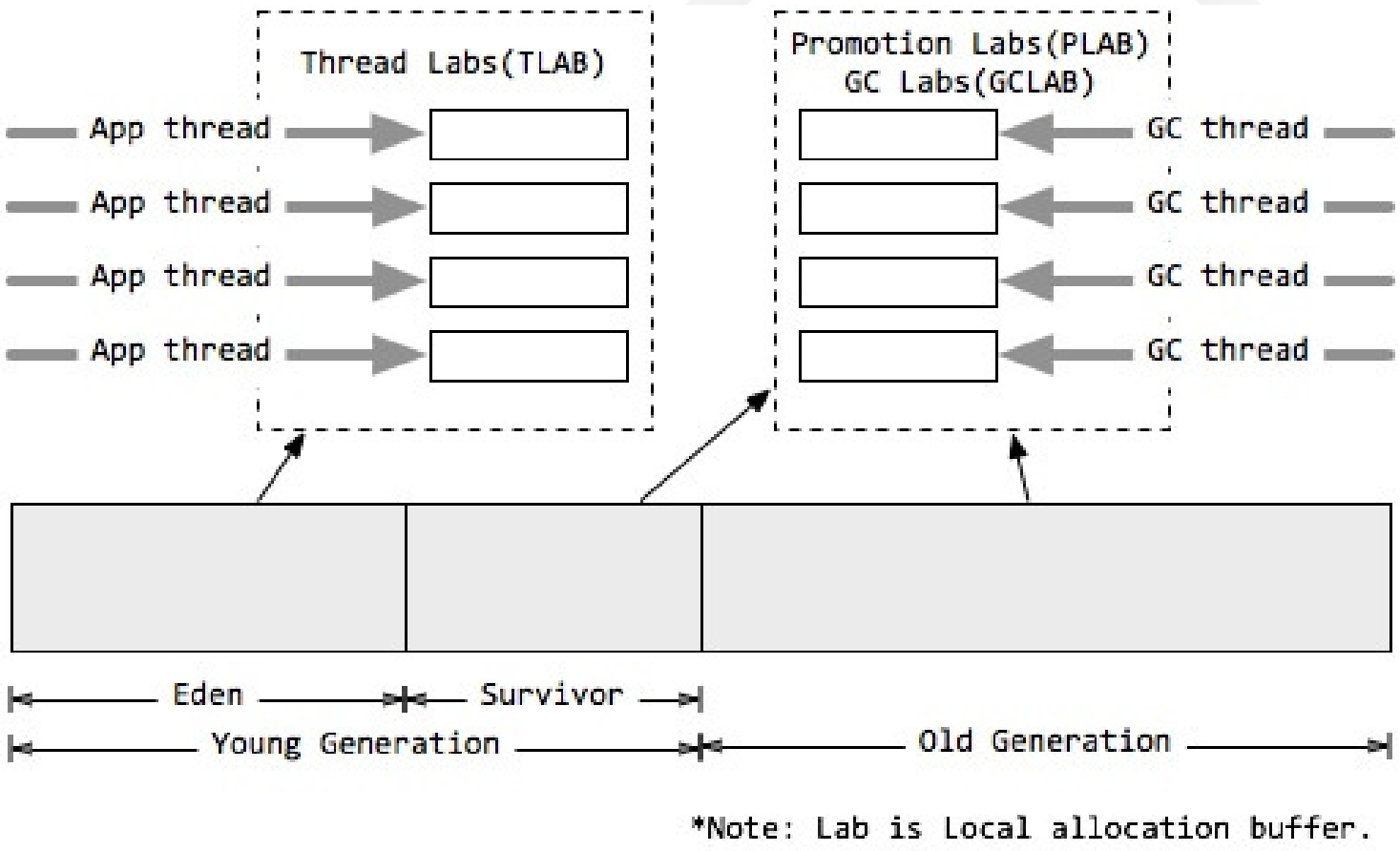
G1 收集器

- G1的内存模型



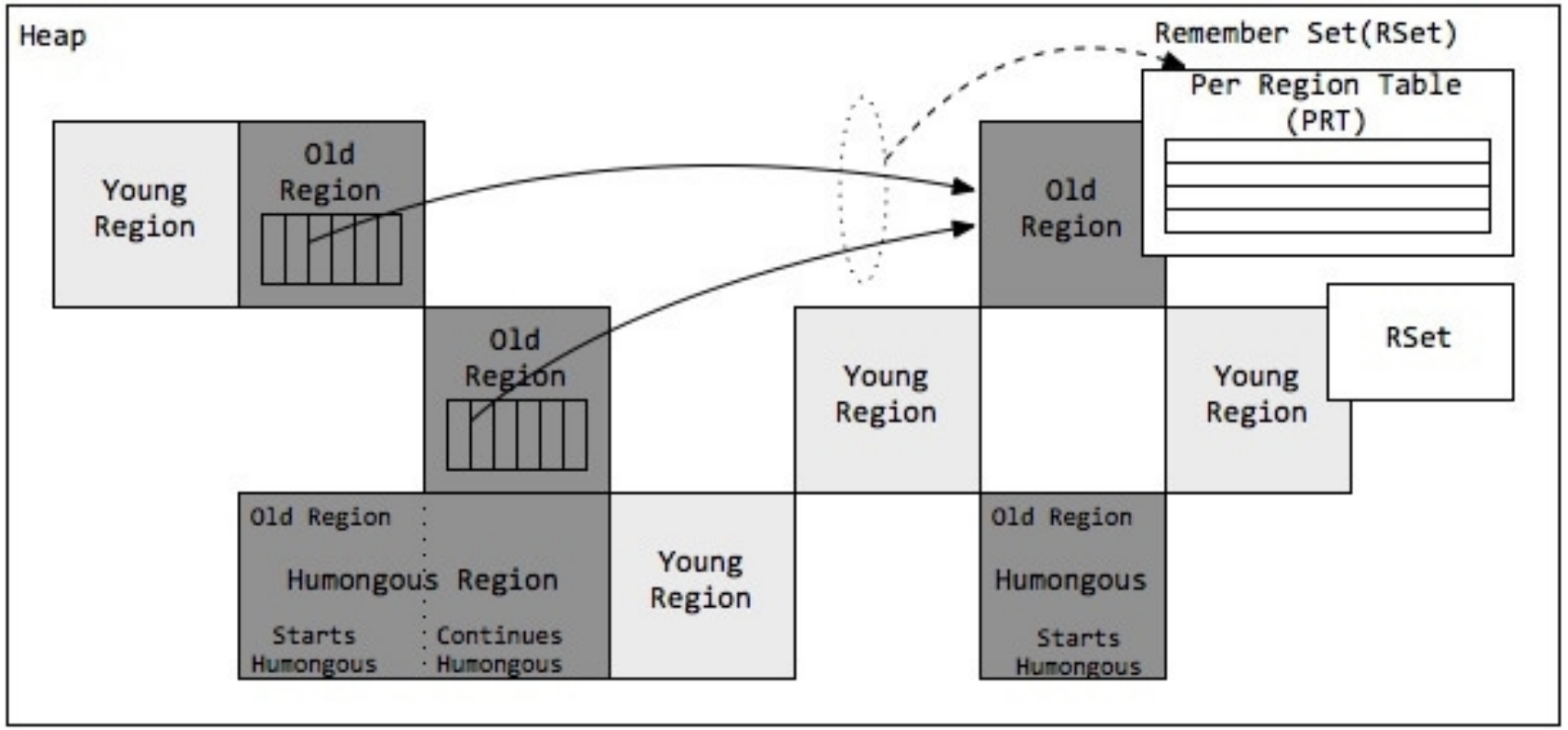
G1 收集器

- G1的分代模型



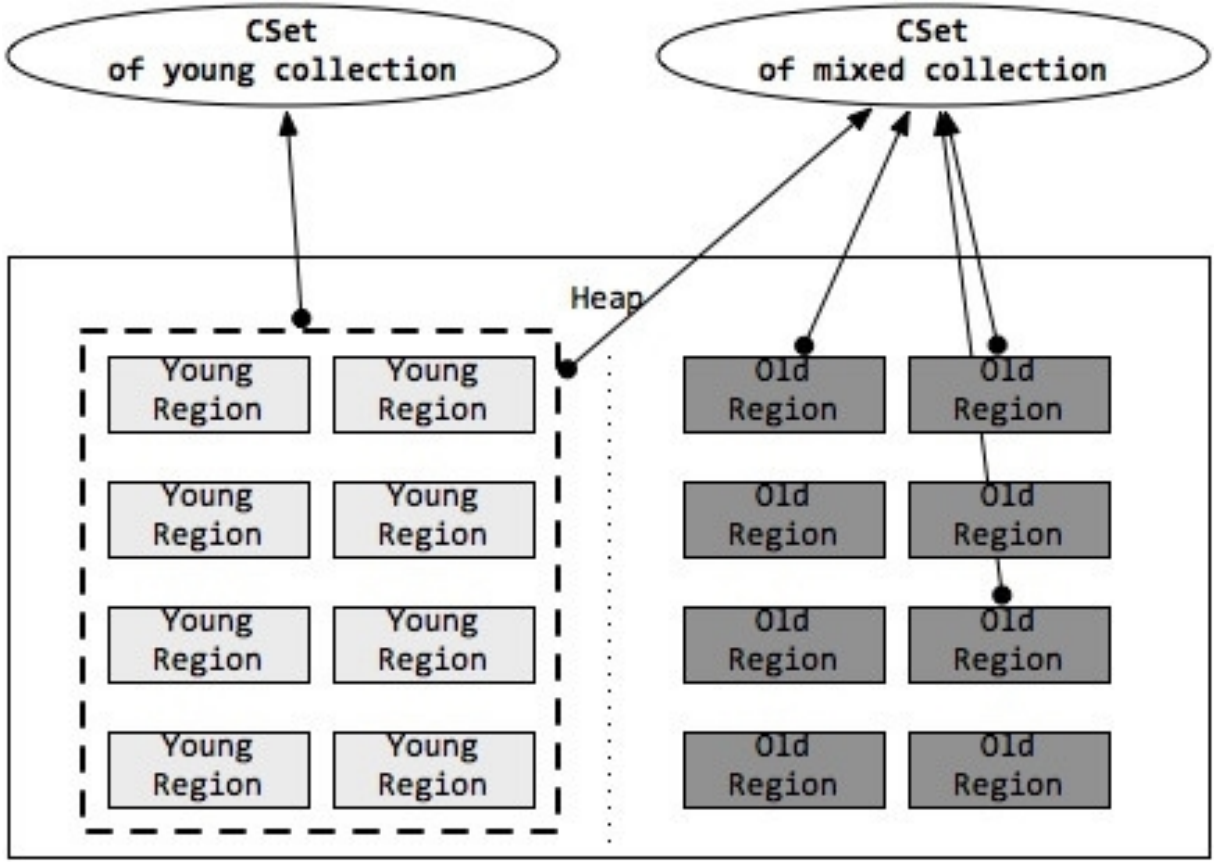
G1 收集器

- G1的分区模型



G1 收集器

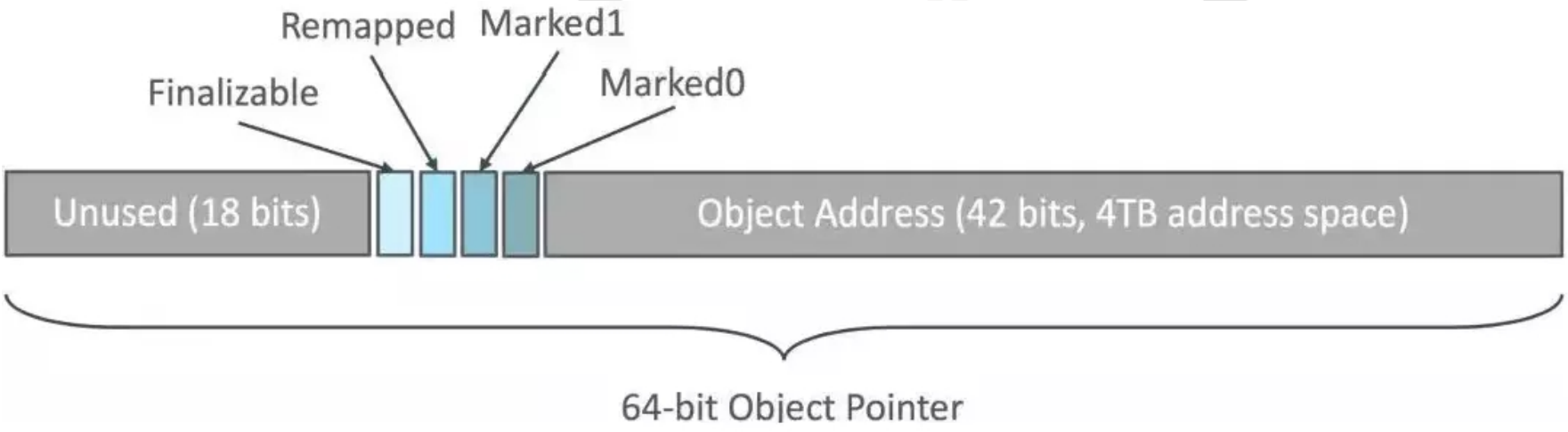
- 收集集合（CSet）
  - 收集集合(CSet)代表每次GC暂停时回收的一系列目标分区



ZGC 收集器

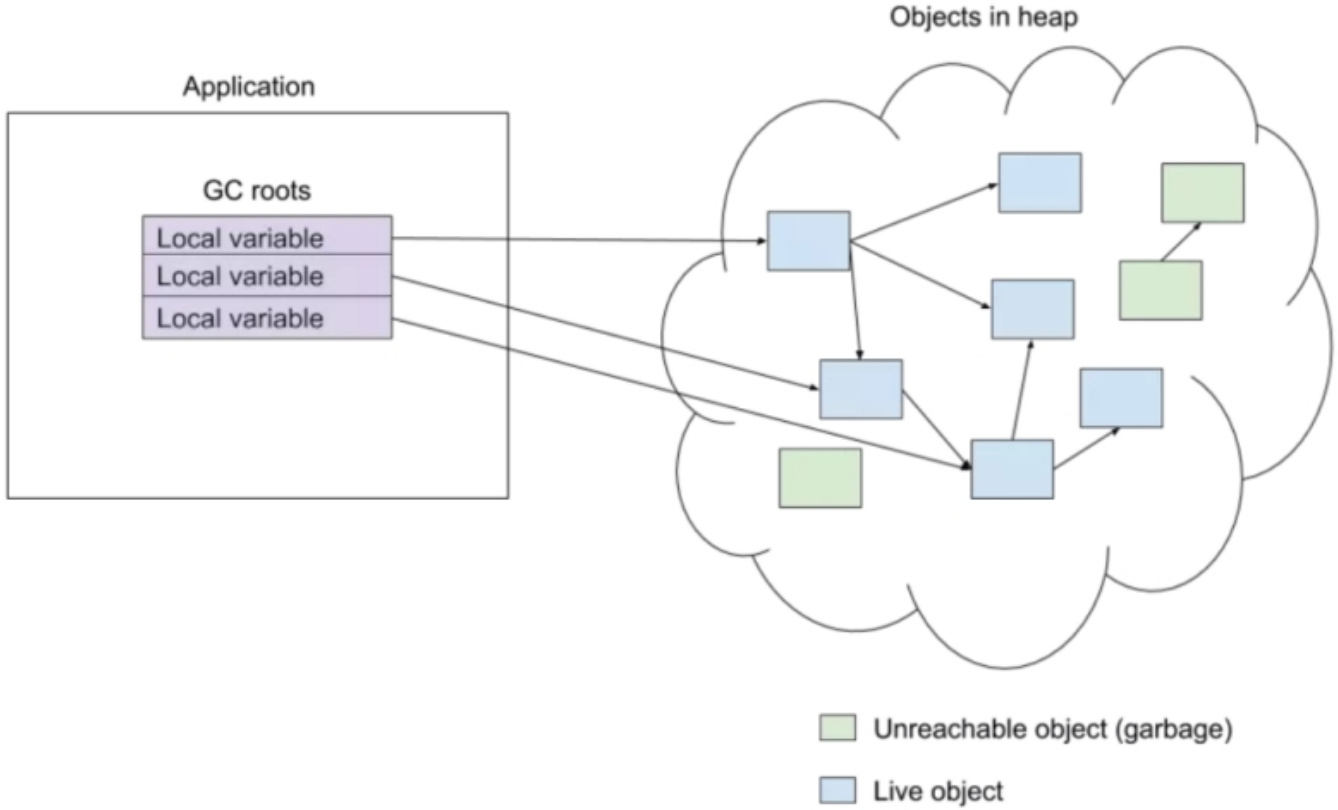
- ZGC原理
  - ZGC在指针上做标记，在访问指针时加入Load Barrier（读屏障），比如当对象正被GC移动，指针上的颜色就会不对，这个屏障就会先把指针更新为有效地址再返回，也就是，永远只有单个对象读取时有概率被减速，而不存在为了保持应用与GC一致而粗暴整体的Stop The World。

- Colored Pointer 和 Load Barrier（并发执行的保证机制）



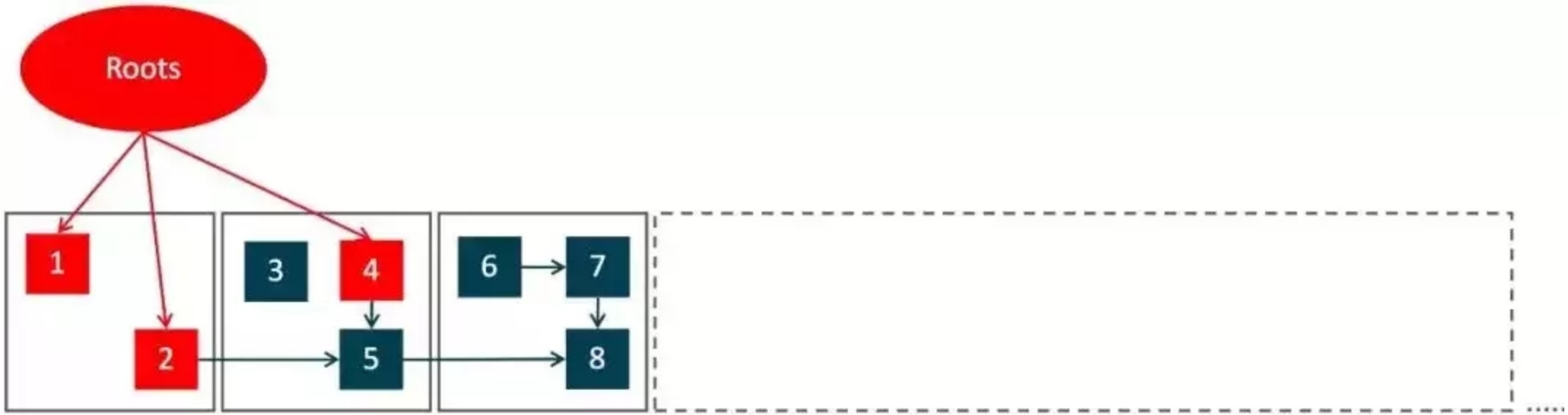
ZGC 内存结构

- ZGC将堆划分为Region作为清理，移动，以及并行GC线程工作分配的单位。分为有2MB，32MB，N× 2MB 三种Size Groups，动态地创建和销毁Region，动态地决定Region的大小。



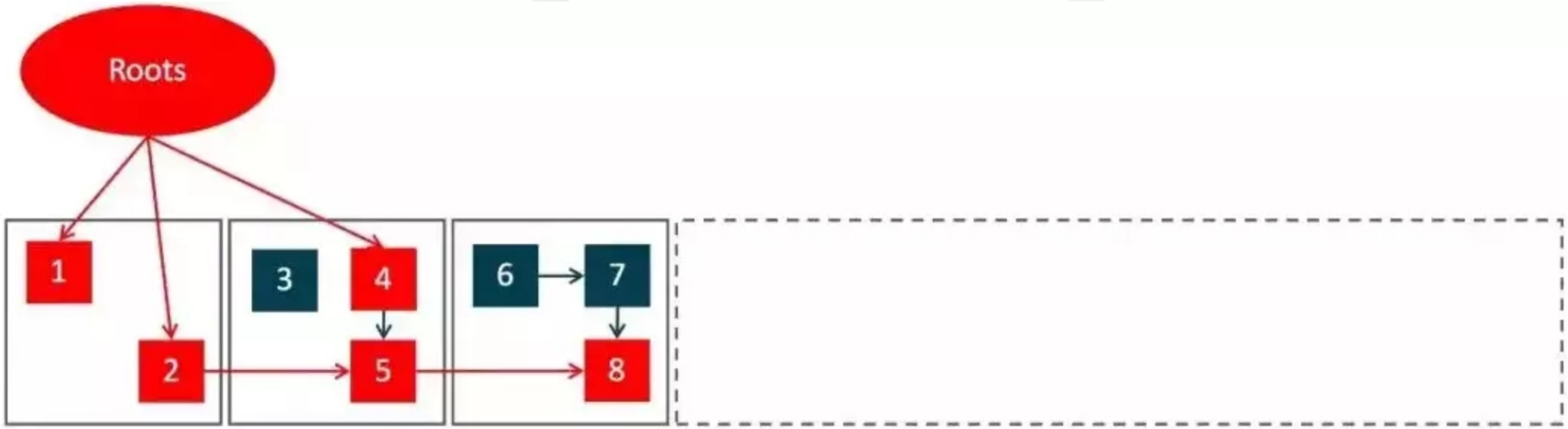
ZGC 回收过程

- Pause Mark Start 一初始停顿标记
  - 停顿JVM，标记Root对象，1，2，4三个被标为live。



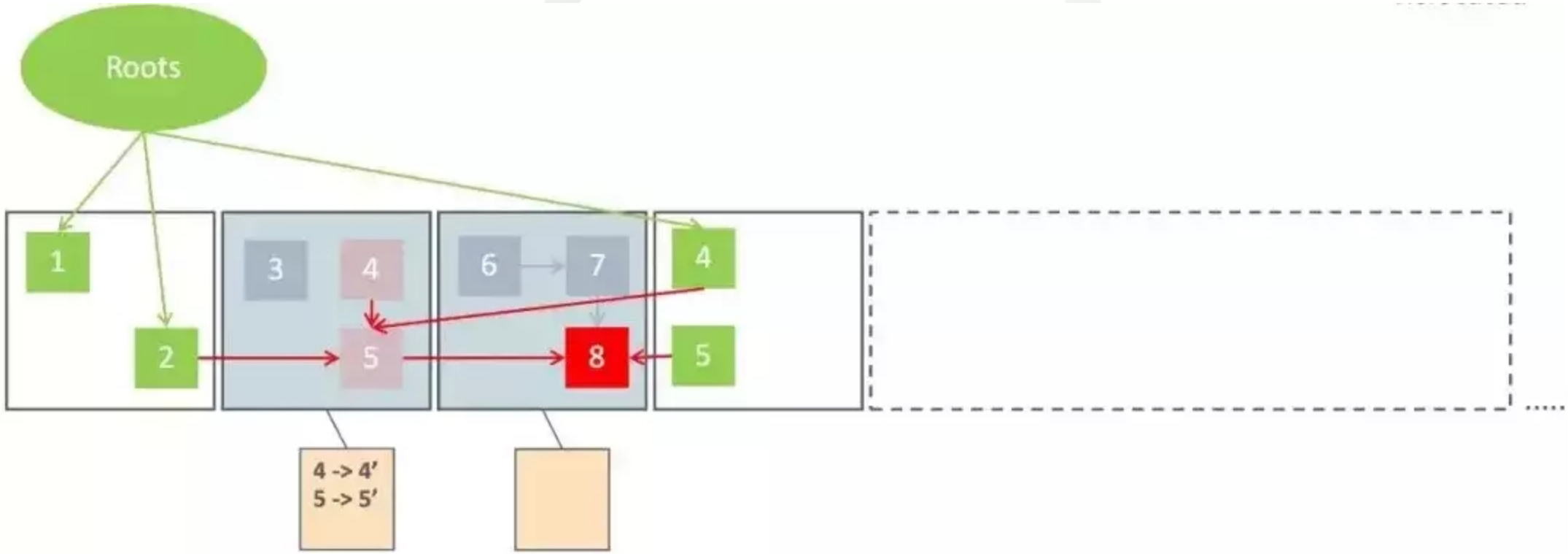
ZGC 回收过程

- Concurrent Mark 一并发标记
  - 并发地递归标记其他对象，5和8也被标记为 live



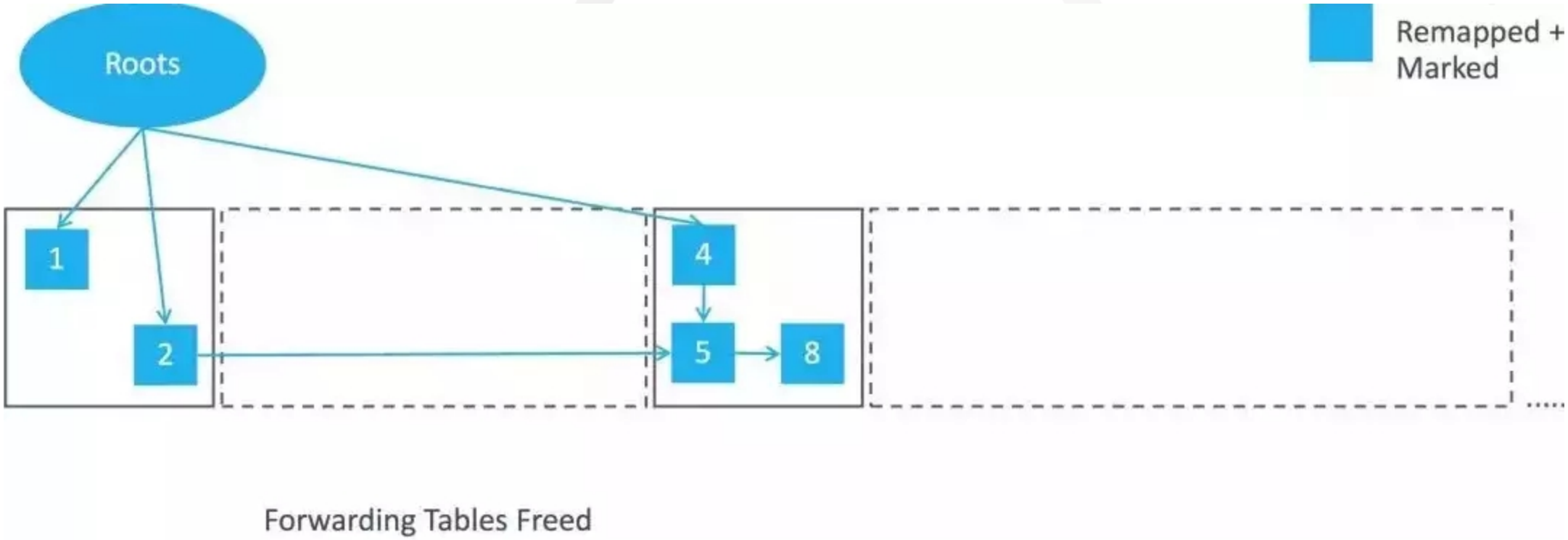
ZGC 回收过程

- Relocate — 移动对象
  - 对比发现3、6、7是过期对象，也就是中间的两个灰色region需要被压缩清理，所以陆续将4、5、8 对象移动到最右边的新Region。移动过程中，有个forward table记录这种转向



ZGC 回收过程

- Remap — 修正指针
  - 最后将指针更新指向新地址。



虚拟机工具

- jps
  - java process status
  - jps -l 主类全名
  - jps -m 运行传入主类的参数
  - jps -v 虚拟机参数
- jstat
  - 类加载，内存，垃圾收集，jit编译信息、
    https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jstat.html
- jinfo
  - 实时调整和查看虚拟机参数
  - -XX:[+/-]option
  - -XX:option=value
- jmap
  - jmap -dump:formart=b,file=filepath pid
  - jmap -histo pid
- jhat
  - JVM heap Analysis Tool
- jstack

- jconsole

专业互联网
IT 教育服务平台

谢 谢 观 看

小白老师QQ号：1828627710