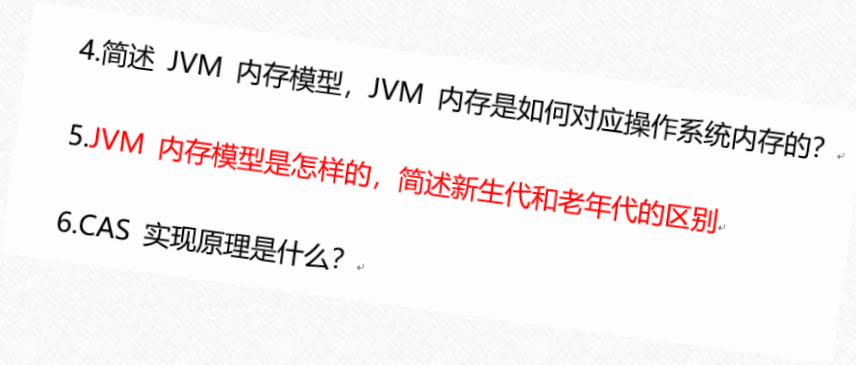
JVM内存模型

罗然 CGB2107

2021.9.27

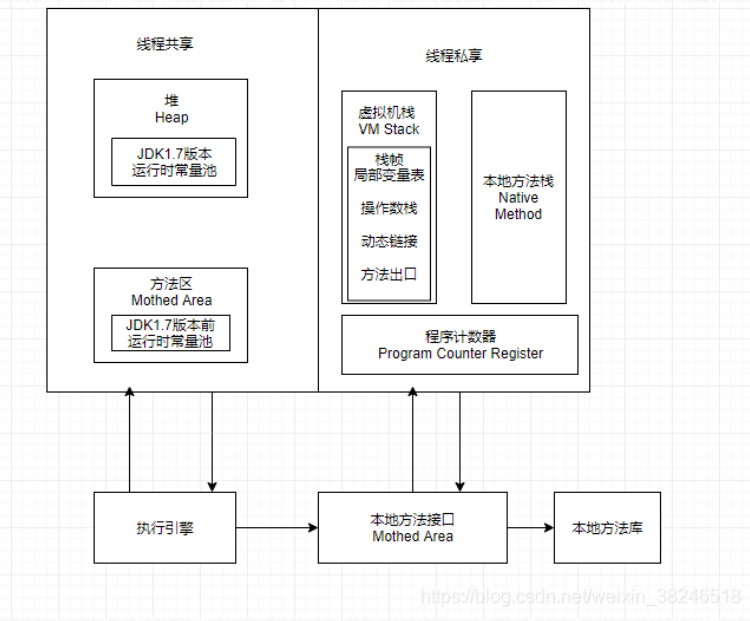
为什么选择该主题？

- 晨讲题型 + 面试重点
- 一阶段总监日拓展内容
- 对内存了解粗浅



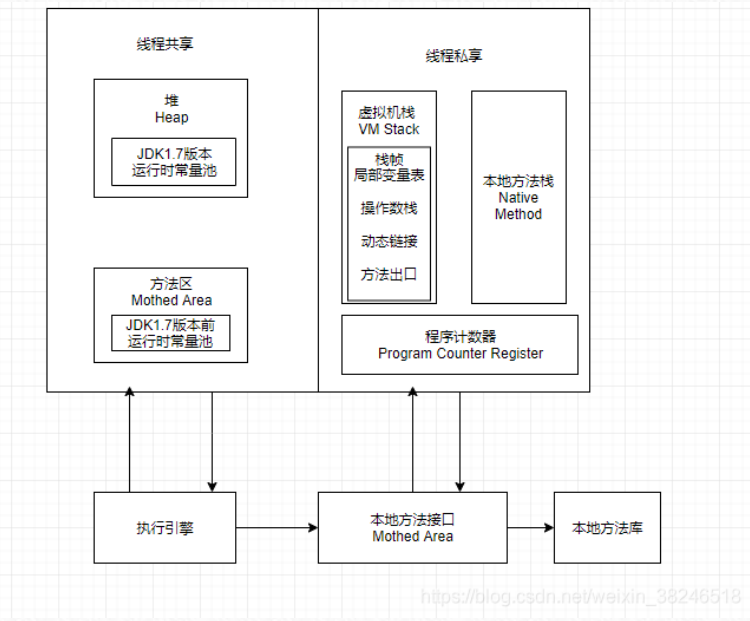
目录

- JVM内存模型概述
- GC垃圾回收机制
- 新生代与老年代

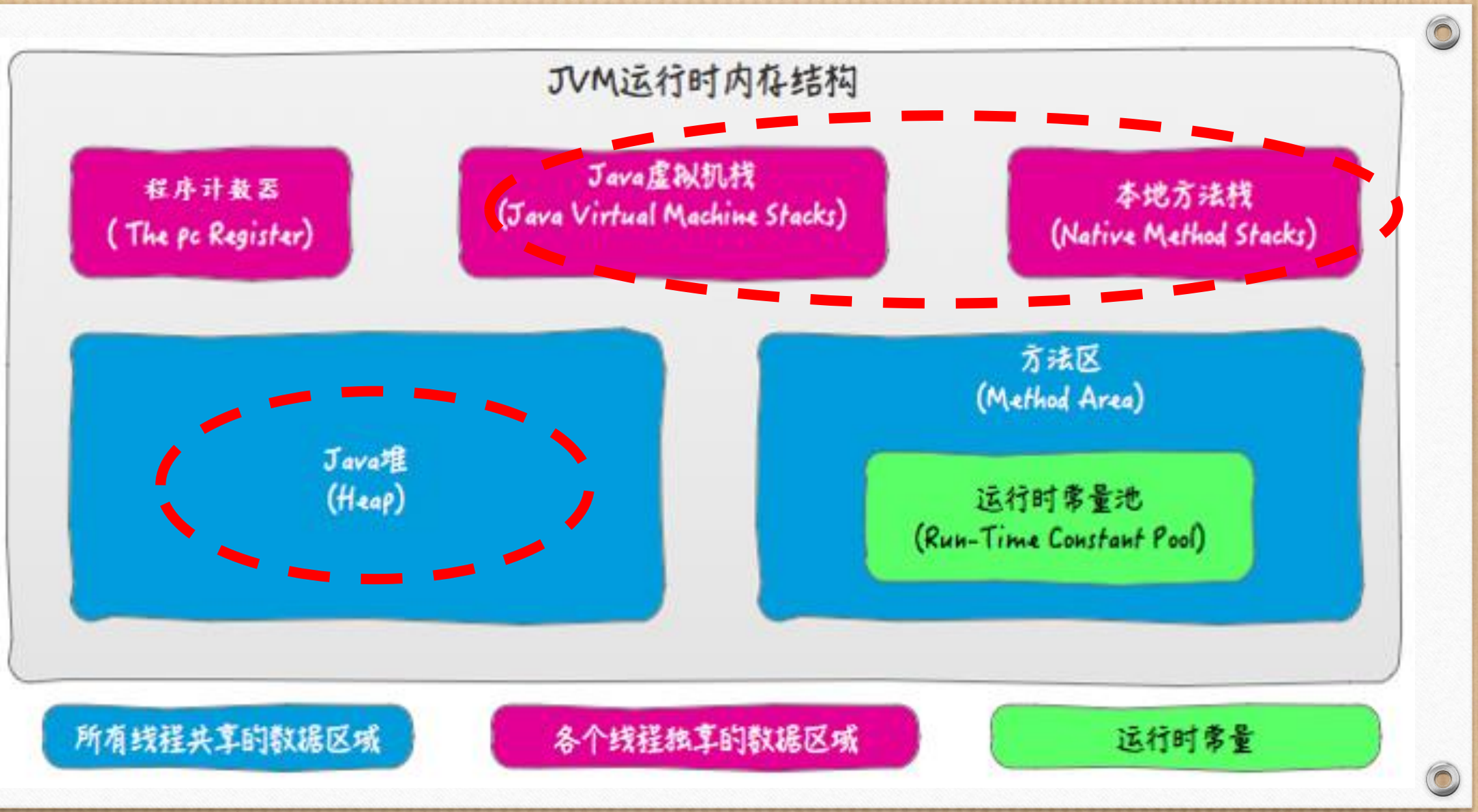


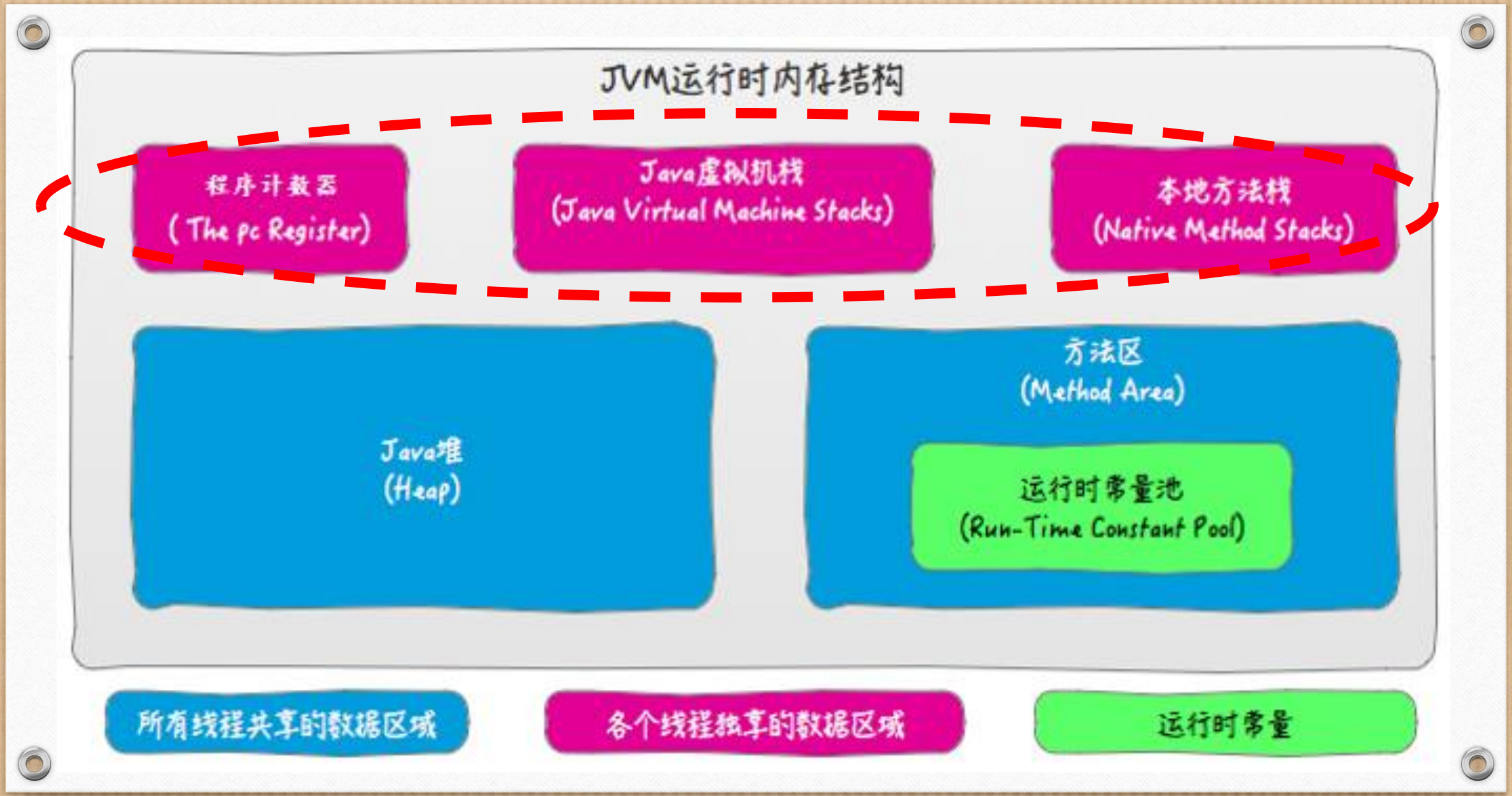
目录

- JVM内存模型概述
- GC垃圾回收机制
- 新生代与老年代




JVM运行时内存结构
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量

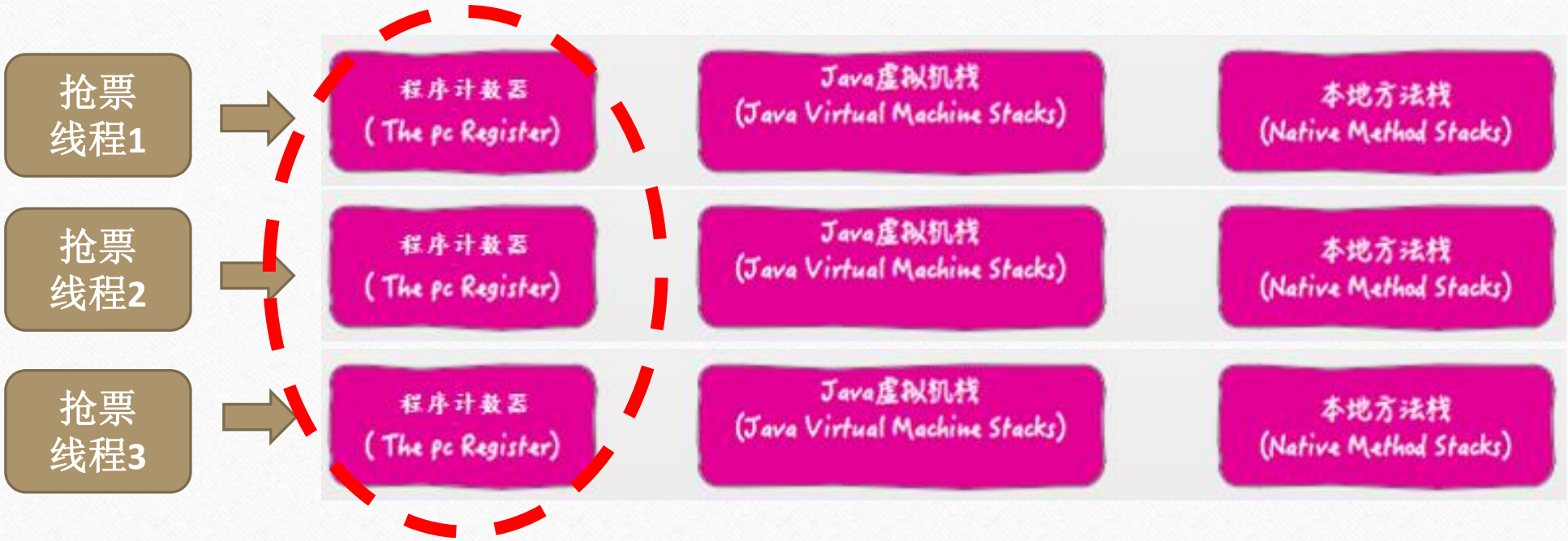

JVM运行时内存结构
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量


JVM运行时内存结构
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量

各个线程独享数据区域

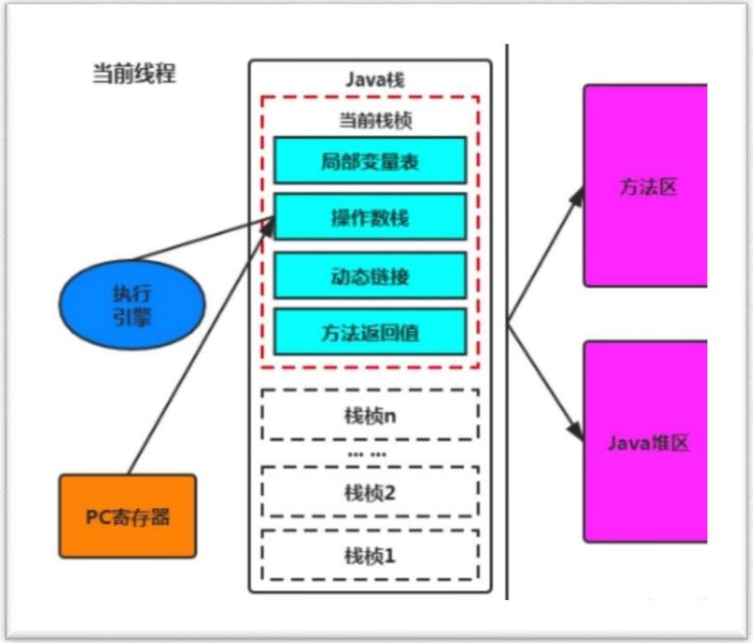

抢票
线程1
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
抢票
线程2
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
抢票
线程3
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)

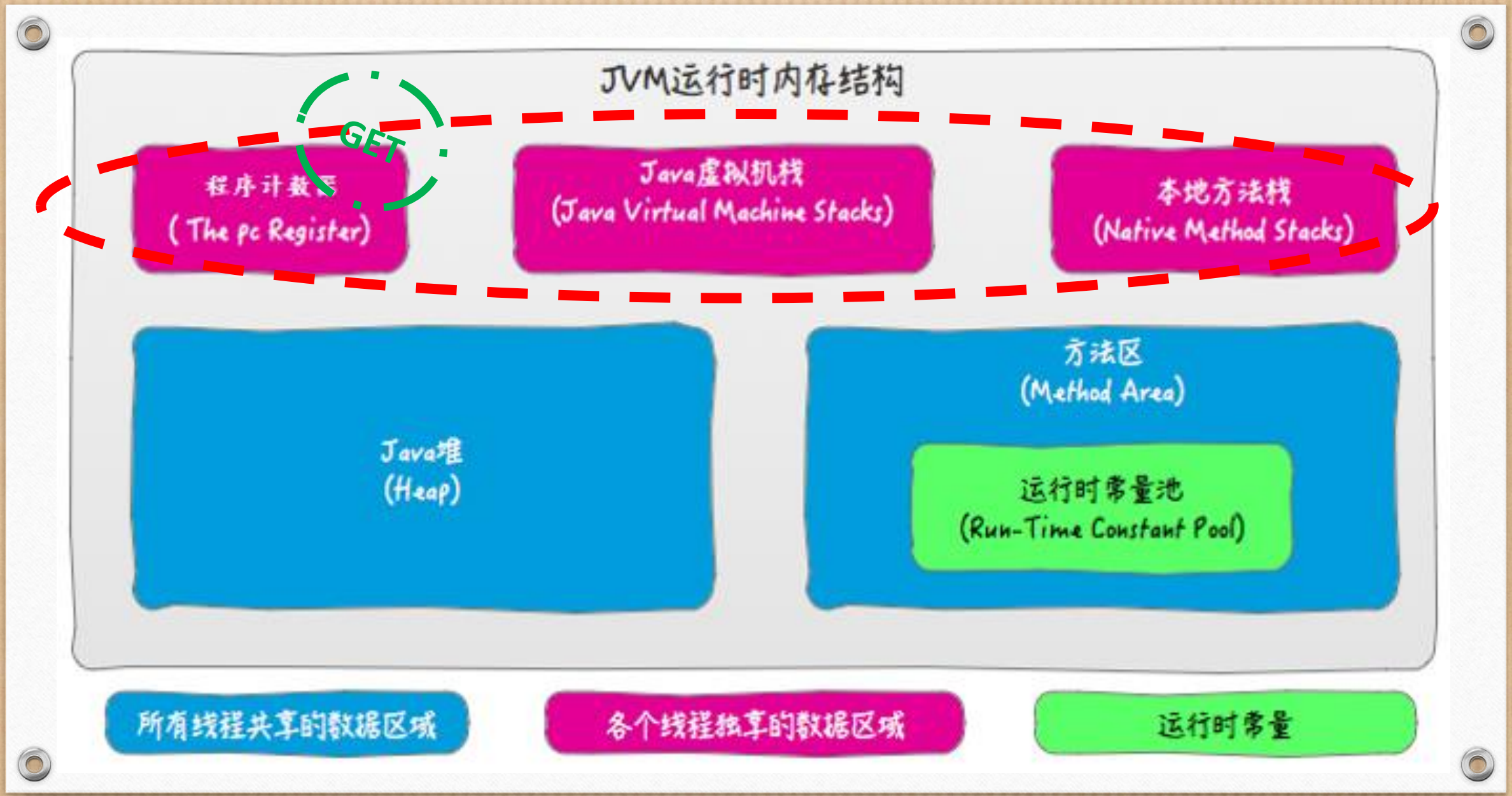
各个线程独享数据区域


抢票
线程1
抢票
线程2
抢票
线程3
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)

程序计数器（Program Counter Register）

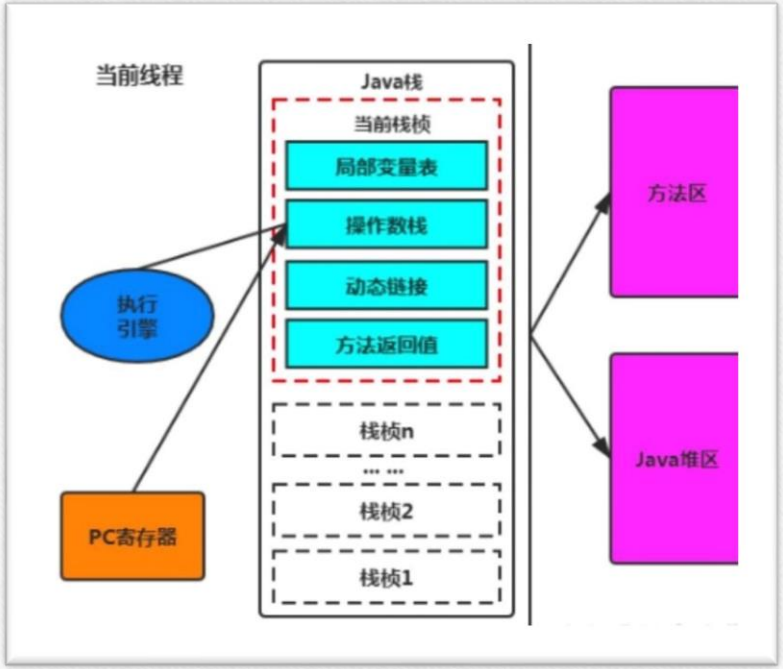
- 是一块较小的内存空间；
- **当前线程所执行字节码**行号指示器，存储指向下一条指令的地址；
- **每条线程都需要有一个独立的程序计数器，**各条线程之间的计数器互不影响，独立存储




JVM运行时内存结构
GET
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量

栈 Stacks

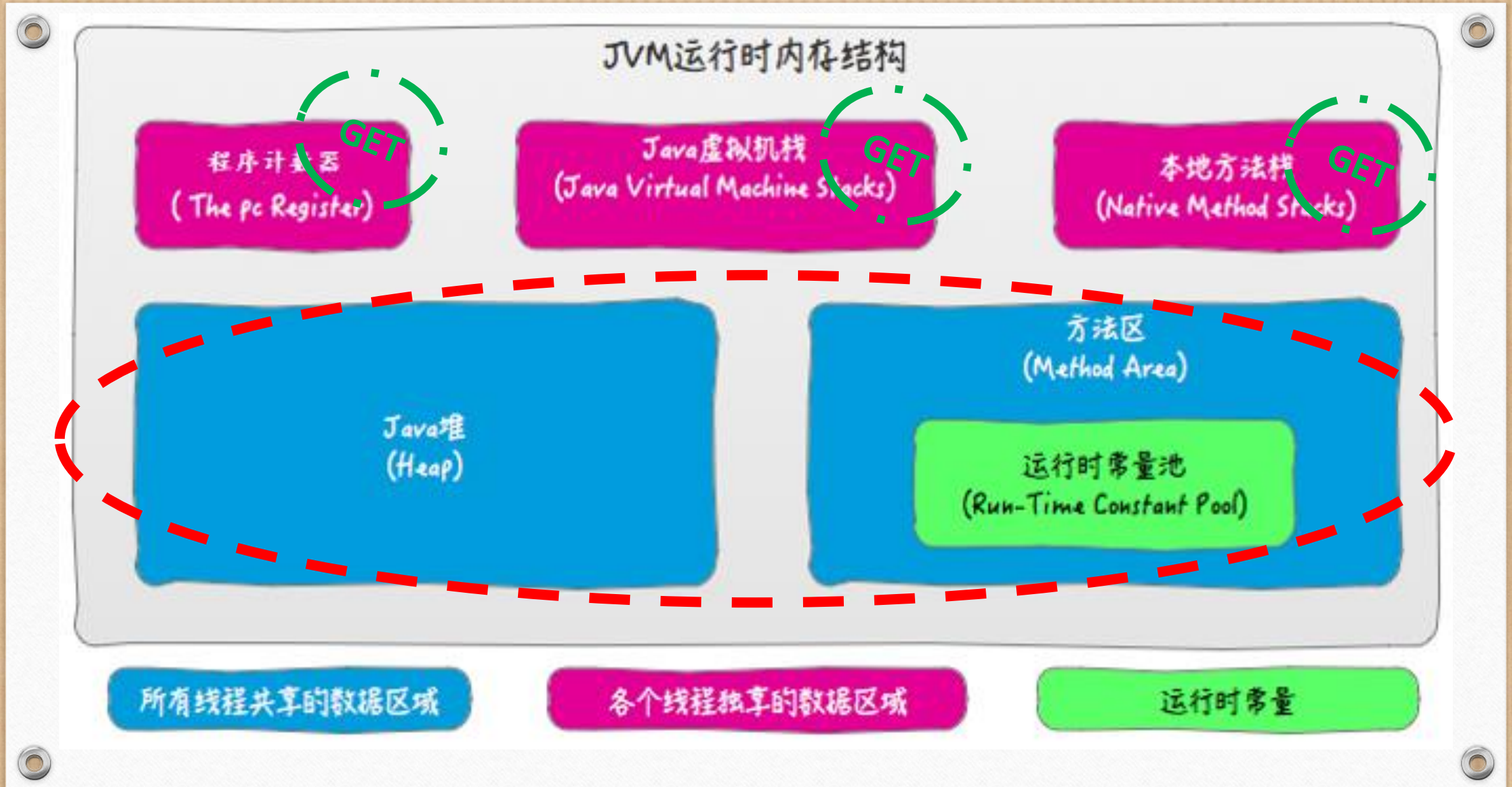
- 虚拟机栈 (Java virtual machine stacks)
  - ✓ 线程里每个方法被执行的时候都会同时创建一个栈帧；
  - ✓ 每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程；



栈 Stacks

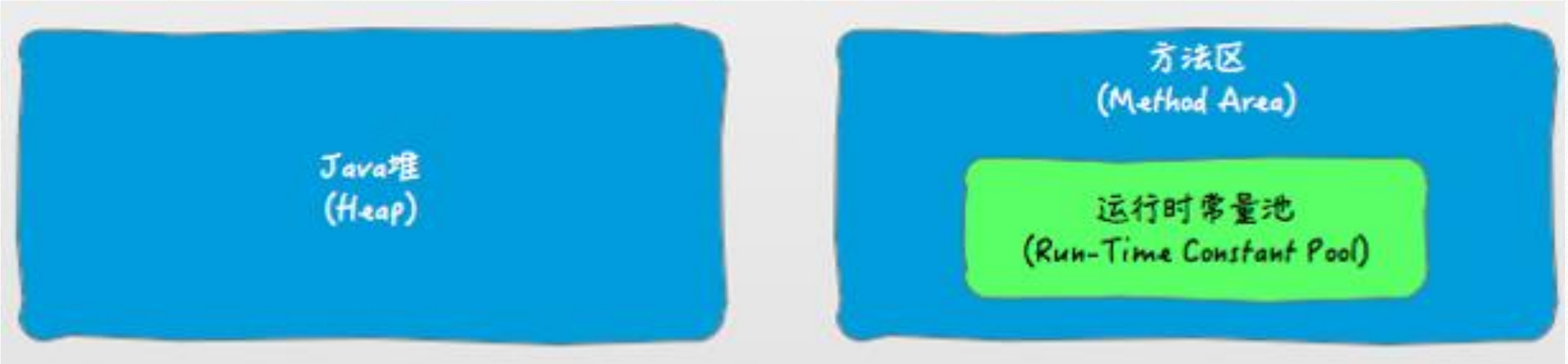
- **本地方法栈 (Native Method Stacks)**
  - ✓ 与虚拟机栈所发挥的作用是类似的；
  - ✓ **区别：**虚拟机栈为执行Java编译后的字节码方法服务，而本地方法栈则是为虚拟机使用到的Native方法服务；

```
class MyClass
{

   static
   {
     System.loadLibrary ("MYSRVPGM");
   }

  native boolean checkCust (byte custName[]);

   void anotherMethod ()
   {
      boolean found;
      // call the native method
      found = checkCust (str.getBytes());
    }
 }
```


JVM运行时内存结构
程序计数器
(The pc Register)
GET
Java虚拟机栈
(Java Virtual Machine Stacks)
GET
本地方法栈
(Native Method Stacks)
GET
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量

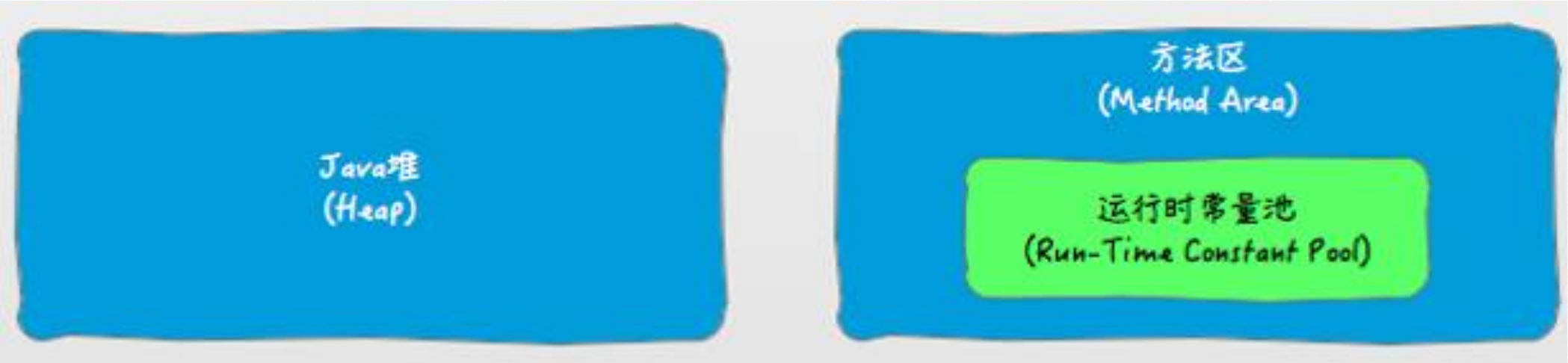
所有线程共享数据区


抢票
线程1
抢票
线程2
抢票
线程4
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)



方法区 (Method Area)

- ✓ 存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码（比如数据字面量）等数据，常量池其实就位于方法区；
- ✓ JVM规范中将方法区描述为堆的逻辑组成，但其别名为 Non-heap —— “我不是堆”，所以应该与”堆”进行区分；



堆 (Heap)

- ✓ 堆是Java 虚拟机所管理的**内存中最大的一块**；
- ✓ Java 堆是被**所有线程共享**的一块内存区域，在虚拟机启动时创建；
- ✓ 此内存区域的**唯一目的就是存放对象实例**，几乎所有的**对象实例都在这里分配内存**；

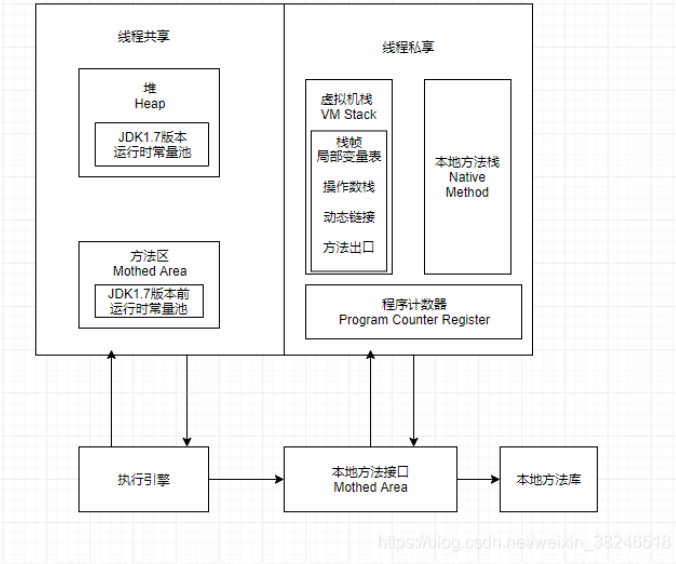

JVM运行时内存结构
程序计数器
(The pc Register)
Java虚拟机栈
(Java Virtual Machine Stacks)
本地方法栈
(Native Method Stacks)
Java堆
(Heap)
方法区
(Method Area)
运行时常量池
(Run-Time Constant Pool)
所有线程共享的数据区域
各个线程独享的数据区域
运行时常量

栈溢出异常是怎么回事？

- 进程每调用一个函数，都会分配一个栈桢，然后在栈桢里会分配函数里定义的各种局部变量，假设现在调用了一个无限递归的函数，那就会持续分配栈帧，但 stack 的大小是有限的（Linux 中默认为 8 M，可以通过 ulimit -a 查看），如果无限递归很快栈就会分配完了，此时再调用函数试图分配超出栈的大小内存，就会发生段错误，也就是 stackOverFlowError

目录

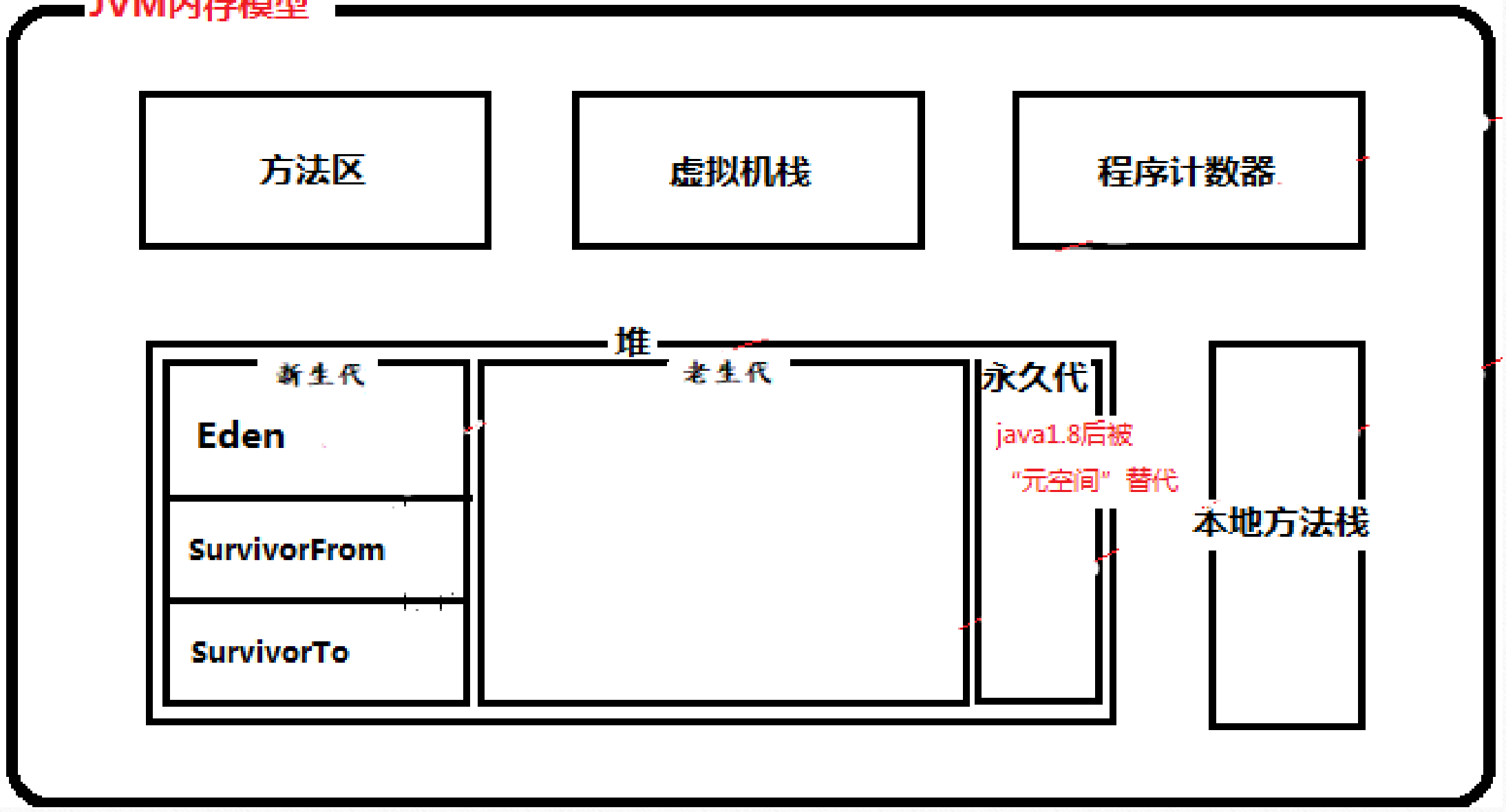
- JVM内存模型概述
- **GC垃圾回收机制**
- 新生代与老年代



GC垃圾回收机制 (Garbage Collection)

- **堆是垃圾收集器管理的主要区域**，所以也被亲切地称为“GC堆”；
- 收集器基本采用**分代收集算法**，所以**堆还可以被细分为**：
  - 新生代 (Eden区 / Survivor-From区 / Survivor- To区)
  - 老年代

JVM内存模型

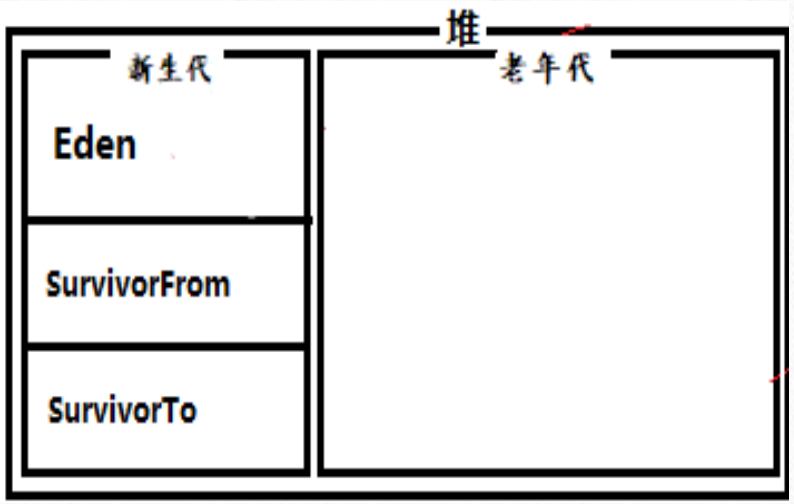

方法区
虚拟机栈
程序计数器
堆
新生代
Eden
SurvivorFrom
SurvivorTo
老生代
永久代
java1.8后被
"元空间"替代
本地方法栈

GC垃圾回收机制

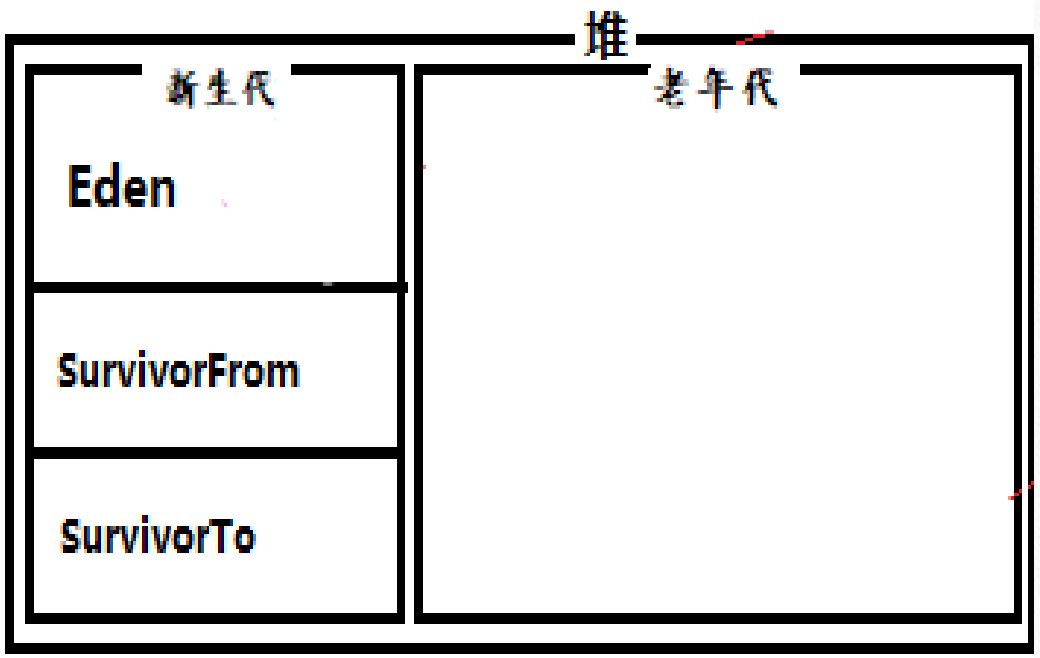
- 将堆分成不同”代”并非必须！

  -> 目的：优化GC性能

- **复制算法：** 将内存划分为两个区间，在任意时间点，**所有动态分配的对象都只能分配在其中一个区间，另一个区间永远是空的**。



新生代GC过程



比率
8:1:1

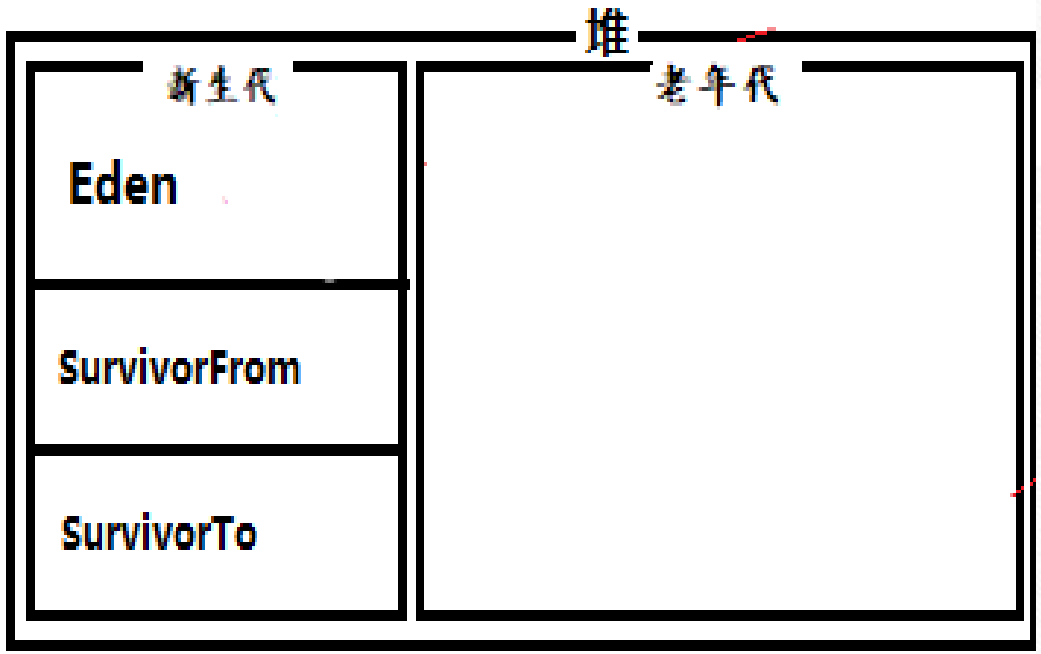
- 实例化新生对象进入Eden区
- Eden满员

Minor GC – 小清理

- 极少数Eden存活下来的对象复制到Survivor From
- 清空Eden
- Eden、Survivor From 满员

Minor GC – 小清理

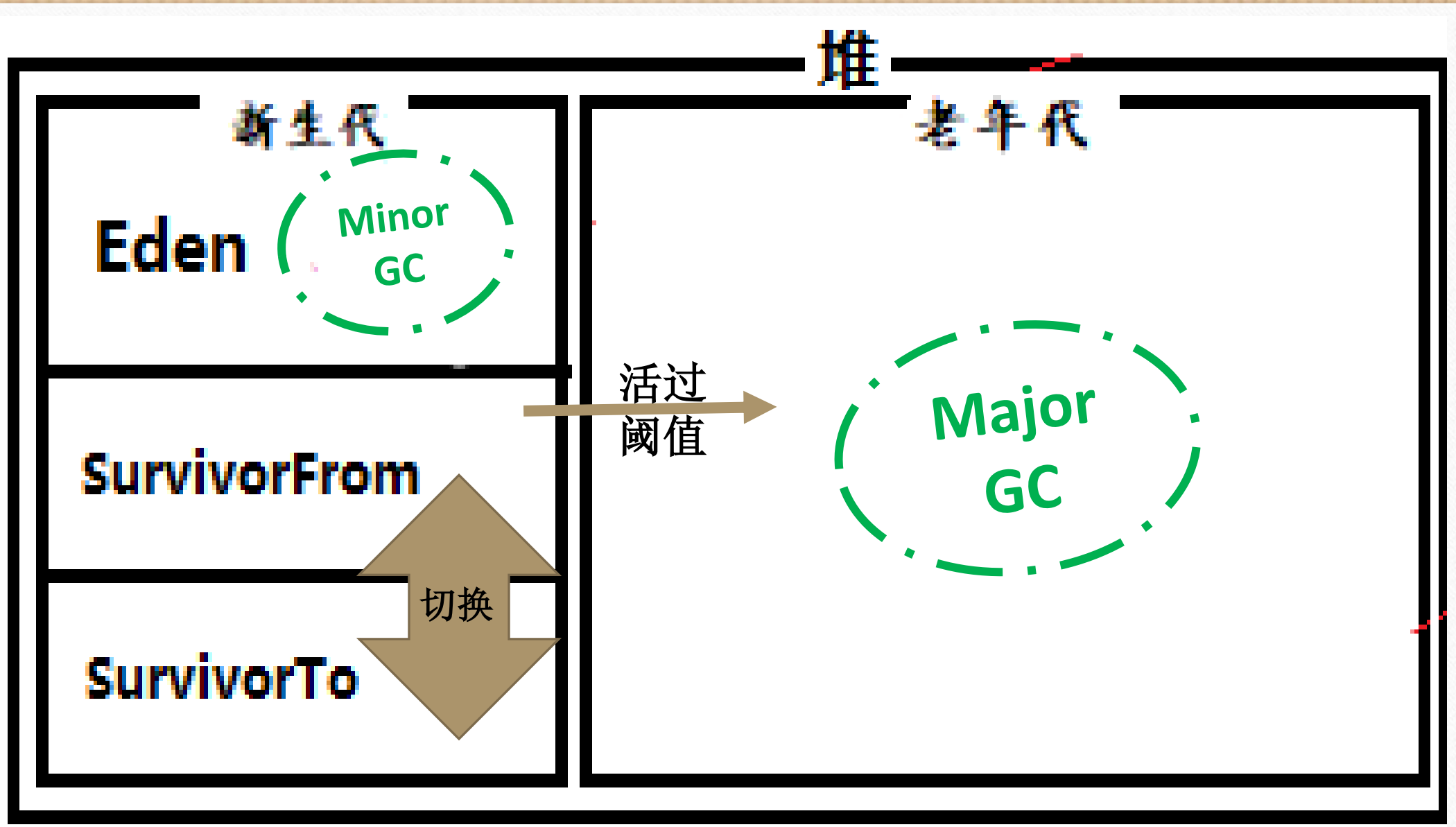
- Eden存活下来的对象复制到Survivor To
- SF存活下来的对象复制到Survivor To
- 清空Eden 、 Survivor From
- Survivor From变为To，To变为From



比率
8:1:1

- 在两个survivor区之间“颠沛流离”活过年龄阈值(比如15)的对象会被送到老年代
- 老年代满员

Major GC – 大清理


堆
新生代
老年代
Eden
Minor
GC
SurvivorFrom
SurvivorTo
切换
活过
阈值
Major
GC


THANK YOU