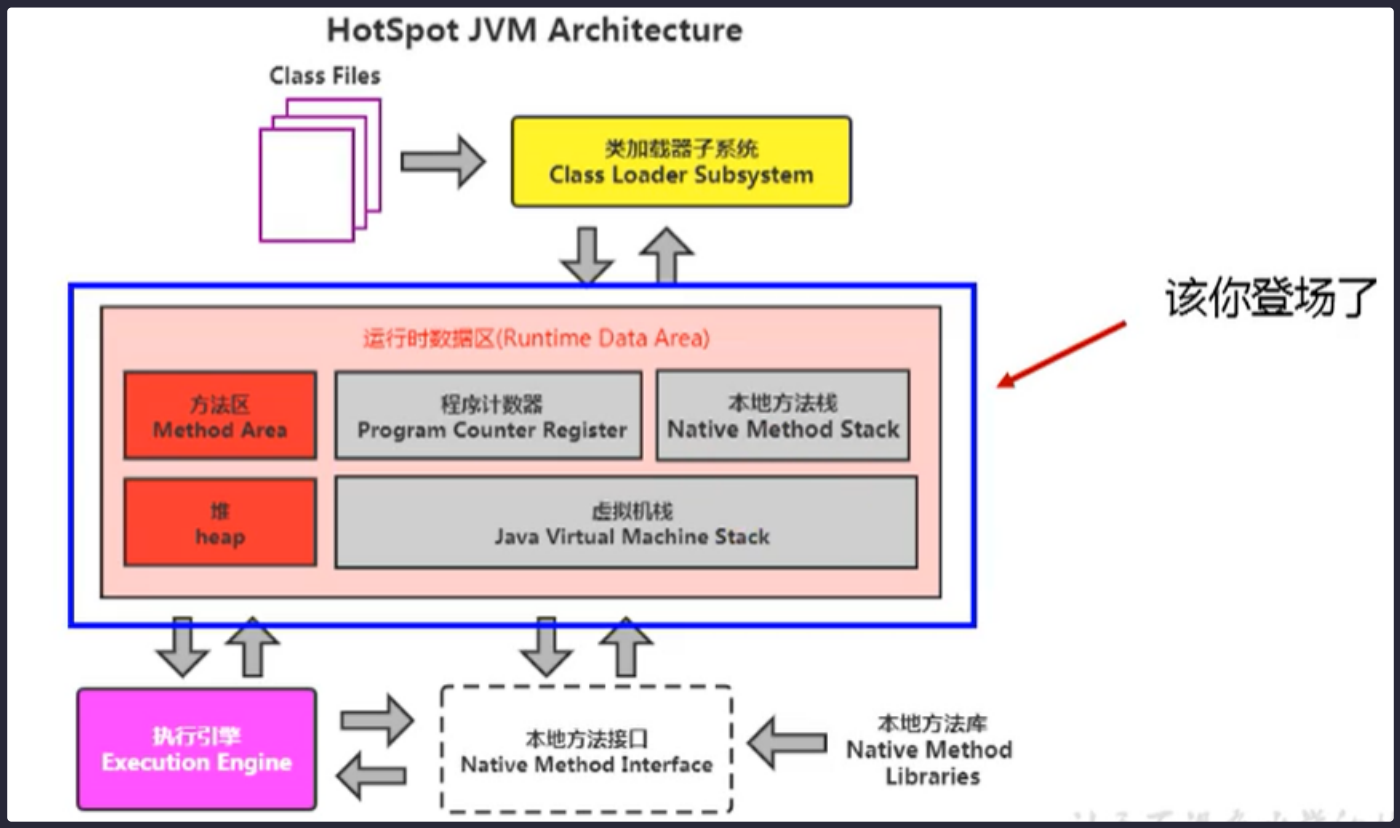
> 笔记来源：尚硅谷 JVM 全套教程，百万播放，全网巅峰（宋红康详解 java 虚拟机）
>
> 同步更新：https://gitee.com/vectorx/NOTE_JVM
>
> https://codechina.csdn.net/qq_35925558/NOTE_JVM
>
> https://github.com/uxiahnan/NOTE_JVM

3. 运行时数据区及程序计数器

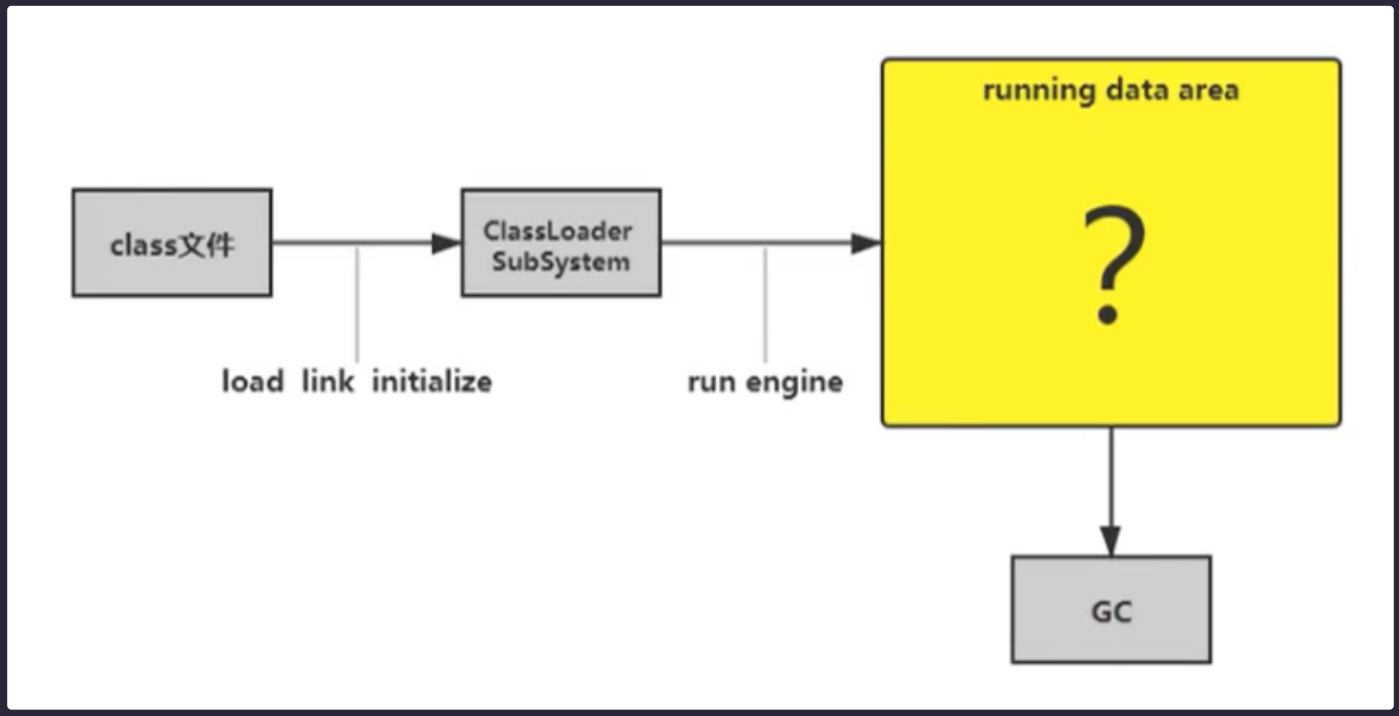
3.1. 运行时数据区

3.1.1. 概述

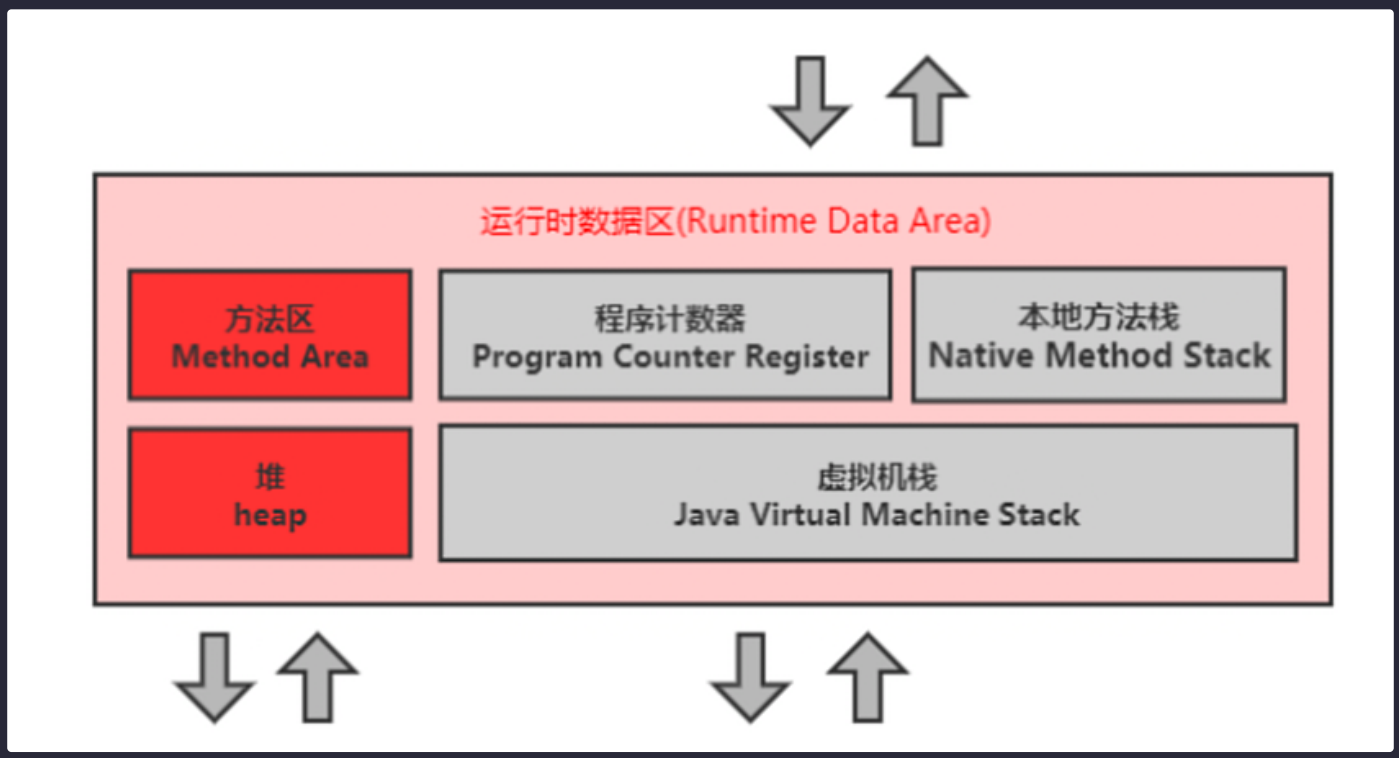
本节主要讲的是运行时数据区，也就是下图这部分，它是在类加载完成后的阶段

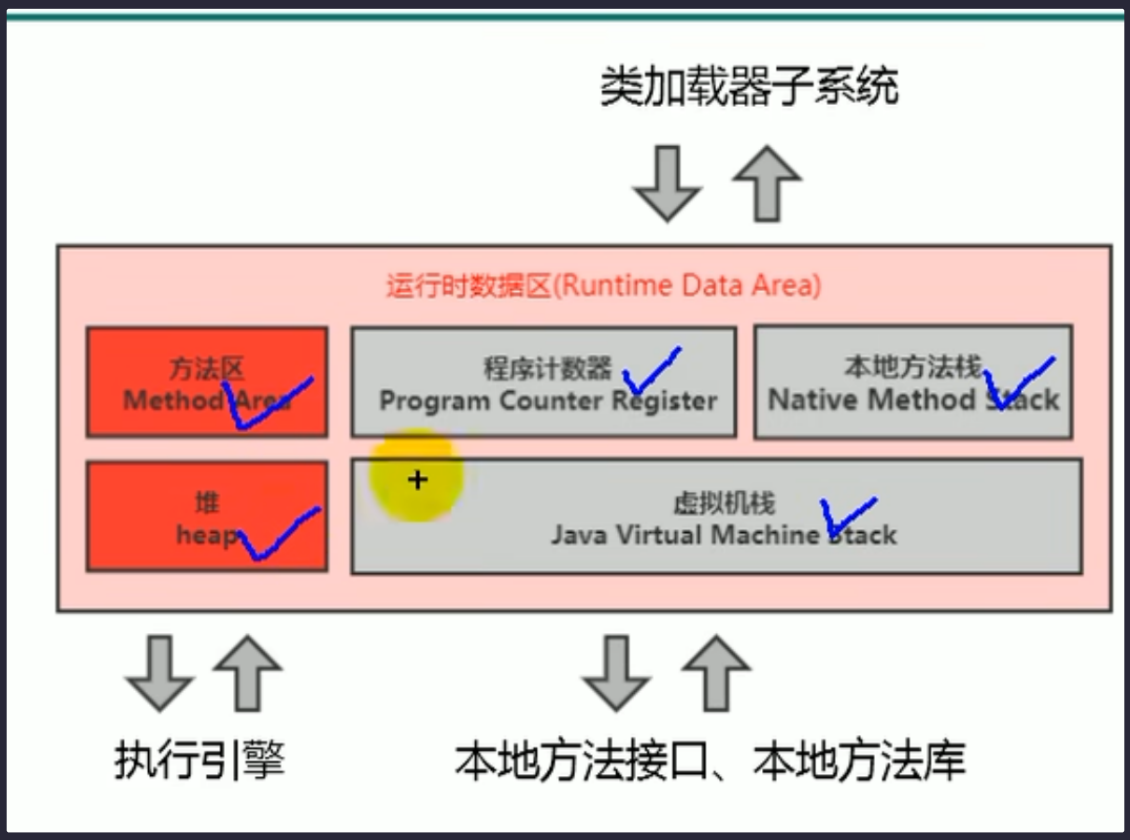


当我们通过前面的：类的加载→ 验证 → 准备 → 解析 → 初始化 这几个阶段完成后，就会用到执行引擎对我们的类进行使用，同时执行引擎将会使用到我们运行时数据区



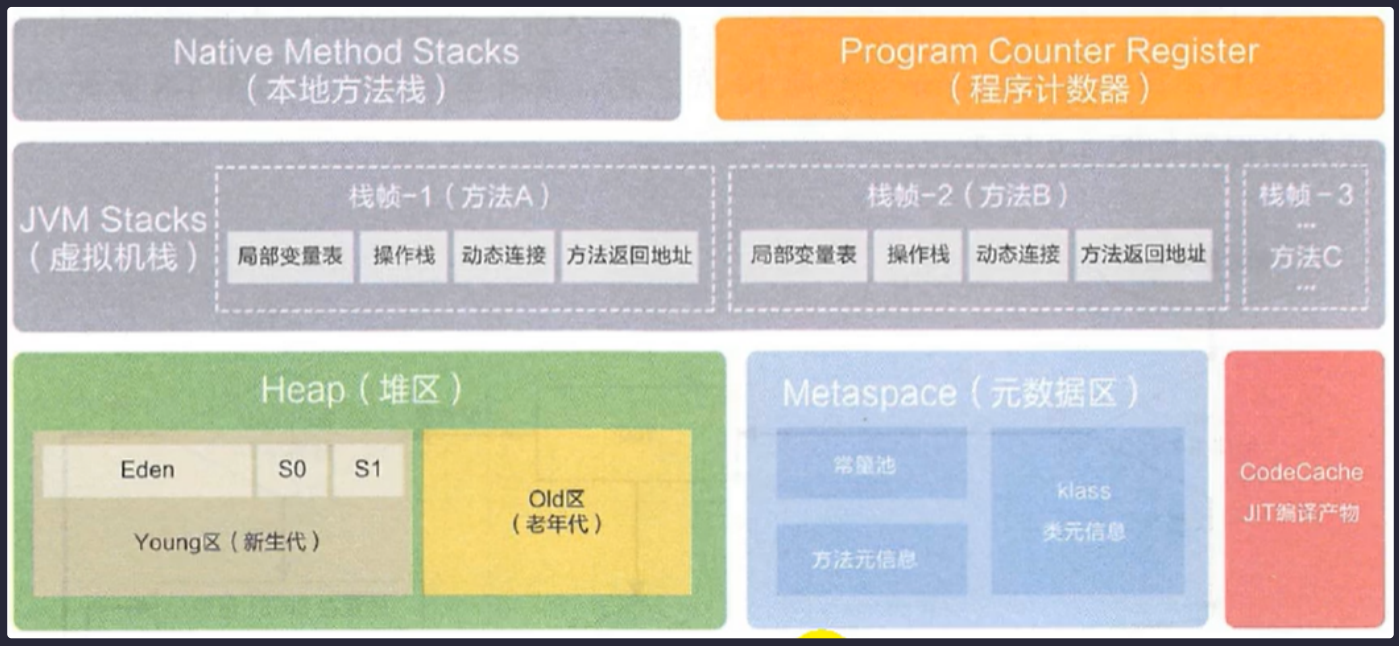
内存是非常重要的系统资源，是硬盘和 CPU 的中间仓库及桥梁，承载着操作系统和应用程序的实时运行 JVM 内存布局规定了 Java 在运行过程中内存申请、分配、管理的策略，保证了 JVM 的高效稳定运行。不同的 JVM 对于内存的划分方式和管理机制存在着部分差异。结合 JVM 虚拟机规范，来探讨一下经典的 JVM 内存布局。





我们把大厨后面的东西（切好的菜，刀，调料），比作是运行时数据区。而厨师可以类比于执行引擎，将通过准备的东西进行制作成精美的菜品

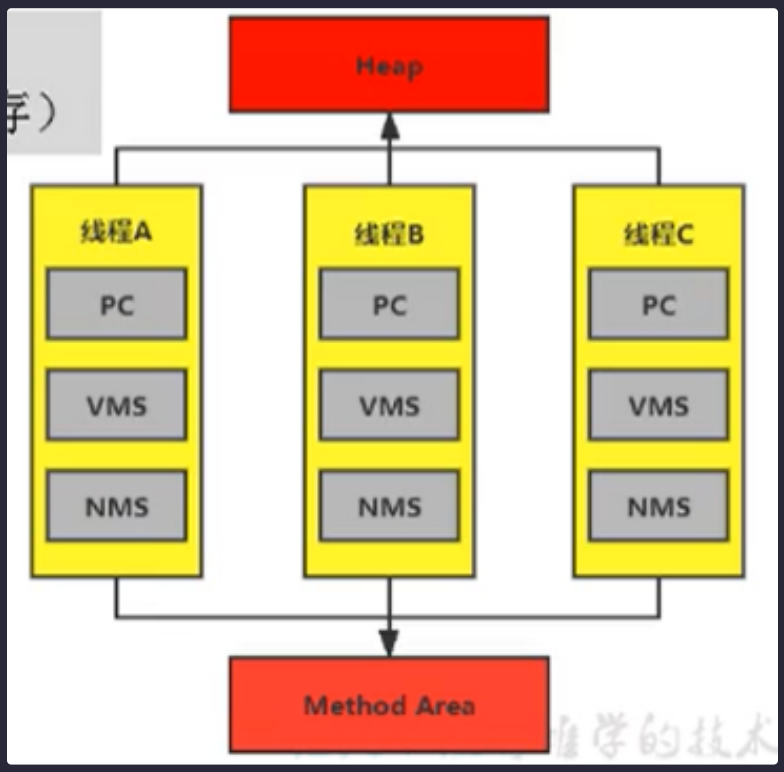
我们通过磁盘或者网络 IO 得到的数据，都需要先加载到内存中，然后 CPU 从内存中获取数据进行读取，也就是说内存充当了 CPU 和磁盘之间的桥梁



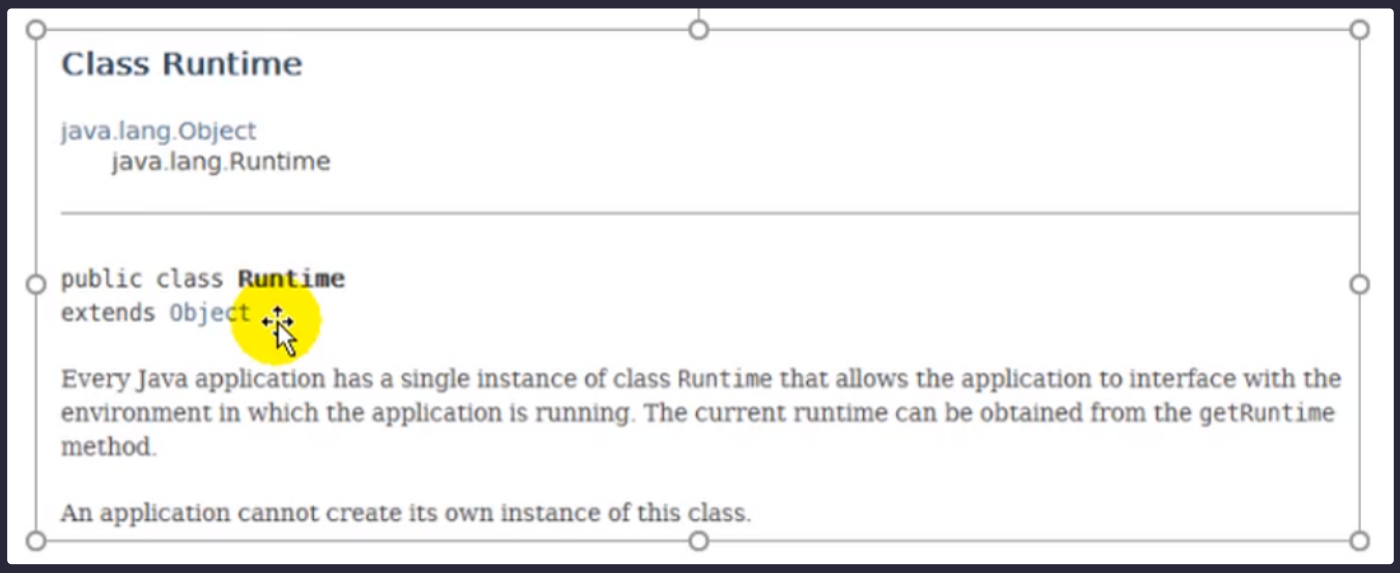
Java 虚拟机定义了若干种程序运行期间会使用到的运行时数据区，其中有一些会随着虚拟机启动而创建，随着虚拟机退出而销毁。另外一些则是与线程一一对应的，这些与线程对应的数据区域会随着线程开始和结束而创建和销毁。

灰色的为单独线程私有的，红色的为多个线程共享的。即：

- 每个线程：独立包括程序计数器、栈、本地栈。
- 线程间共享：堆、堆外内存（永久代或元空间、代码缓存）



每个 JVM 只有一个 Runtime 实例。即为运行时环境，相当于内存结构的中间的那个框框：运行时环境。



3.1.2. 线程

线程是一个程序里的运行单元。JVM 允许一个应用有多个线程并行的执行。 在 Hotspot JVM 里，每个线程都与操作系统的本地线程直接映射 。

当一个 Java 线程准备好执行以后，此时一个操作系统的本地线程也同时创建。Java 线程执行终止后，本地线程也会回收。

操作系统负责所有线程的安排调度到任何一个可用的 CPU 上。一旦本地线程初始化成功，它就会调用 Java 线程中的 run()方法。

线程分为 守护线程 和 普通线程。如果仅剩守护线程，那么Java程序也就该结束了。

3.1.3. JVM 系统线程

如果你使用 jconsole 或者是任何一个调试工具，都能看到在后台有许多线程在运行。这些后台线程不包括调用 `public static void main(String[] args)` 的 main 线程以及所有这个 main 线程自己创建的线程。

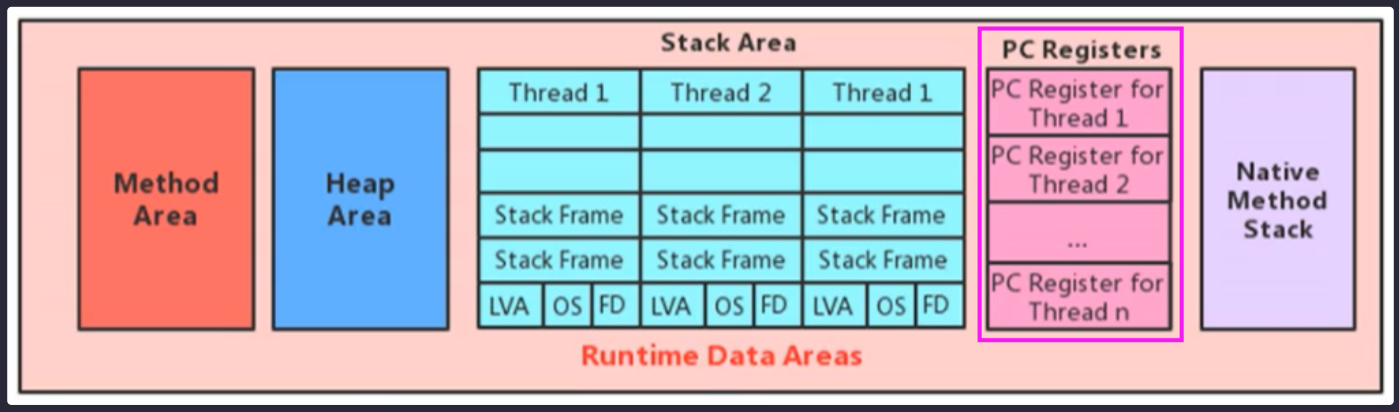
这些主要的后台系统线程在 Hotspot JVM 里主要是以下几个：

- 虚拟机线程：这种线程的操作是需要 JVM 达到安全点才会出现。这些操作必须在不同的线程中发生的原因是他们都需要 JVM 达到安全点，这样堆才不会变化。这种线程的执行类型包括"stop-the-world"的垃圾收集，线程栈收集，线程挂起以及偏向锁撤销。
- 周期任务线程：这种线程是时间周期事件的体现（比如中断），他们一般用于周期性操作的调度执行。
- GC 线程：这种线程对在 JVM 里不同种类的垃圾收集行为提供了支持。
- 编译线程：这种线程在运行时会将字节码编译成到本地代码。
- 信号调度线程：这种线程接收信号并发送给 JVM，在它内部通过调用适当的方法进行处理。

3.2. 程序计数器(PC 寄存器)

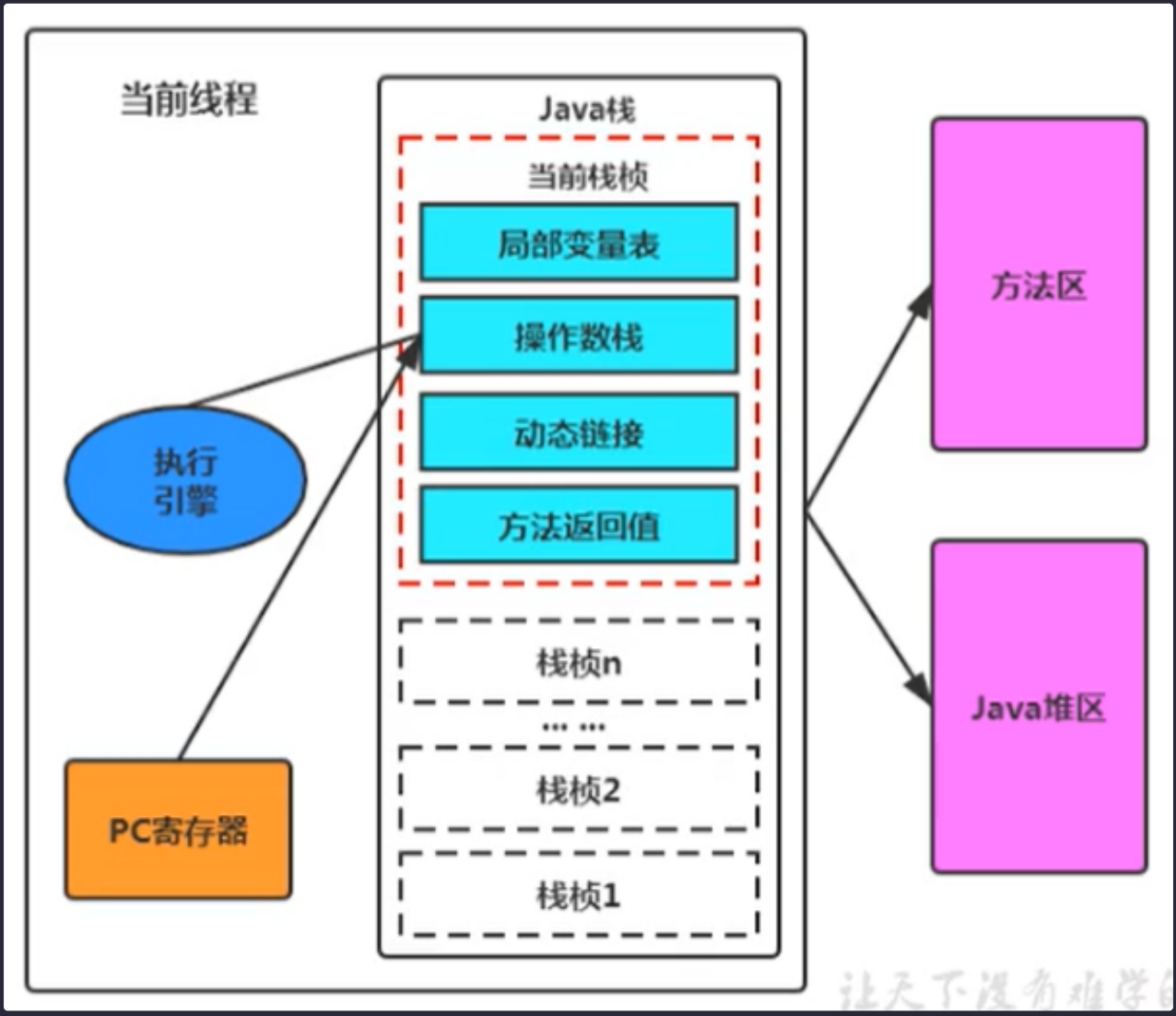
JVM 中的程序计数寄存器（Program Counter Register）中，Register 的命名源于 CPU 的寄存器，寄存器存储指令相关的现场信息。CPU 只有把数据装载到寄存器才能够运行。

这里，并非是广义上所指的物理寄存器，或许将其翻译为 PC 计数器（或指令计数器）会更加贴切（也称为程序钩子），并且也不容易引起一些不必要的误会。JVM 中的 PC 寄存器是对物理 PC 寄存器的一种抽象模拟。



作用

PC 寄存器用来存储指向下一条指令的地址，也即将要执行的指令代码。由执行引擎读取下一条指令。



它是一块很小的内存空间，几乎可以忽略不记。也是运行速度最快的存储区域。

在 JVM 规范中，每个线程都有它自己的程序计数器，是线程私有的，生命周期与线程的生命周期保持一致。

任何时间一个线程都只有一个方法在执行，也就是所谓的当前方法。程序计数器会存储当前线程正在执行的 Java 方法的 JVM 指令地址；或者，如果是在执行 native 方法，则是未指定值（undefined）。

它是程序控制流的指示器，分支、循环、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器来完成。

字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令。

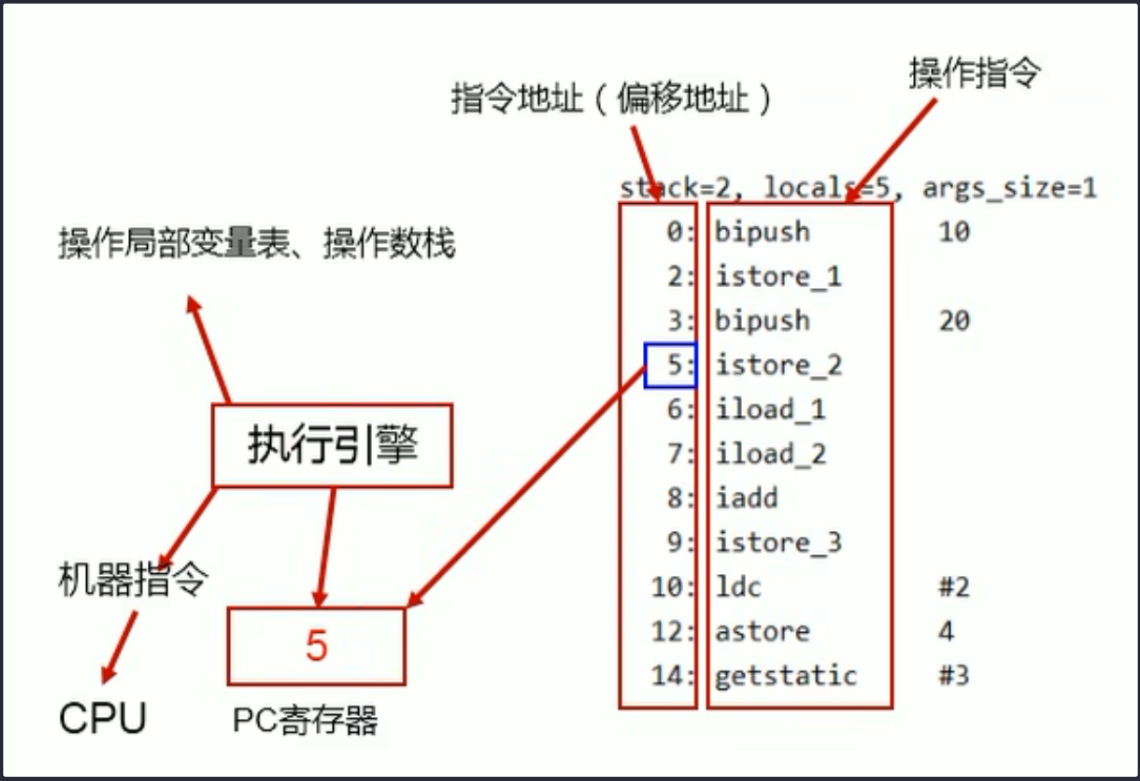
它是唯一一个在 Java 虚拟机规范中没有规定任何 OutofMemoryError 情况的区域（也没有垃圾回收）。

举例说明

```
public int minus() {
    intc = 3;
    intd = 4;
    return c - d;
}
```

字节码文件：（左边数字:指令地址/偏移地址　右边:操作指令）

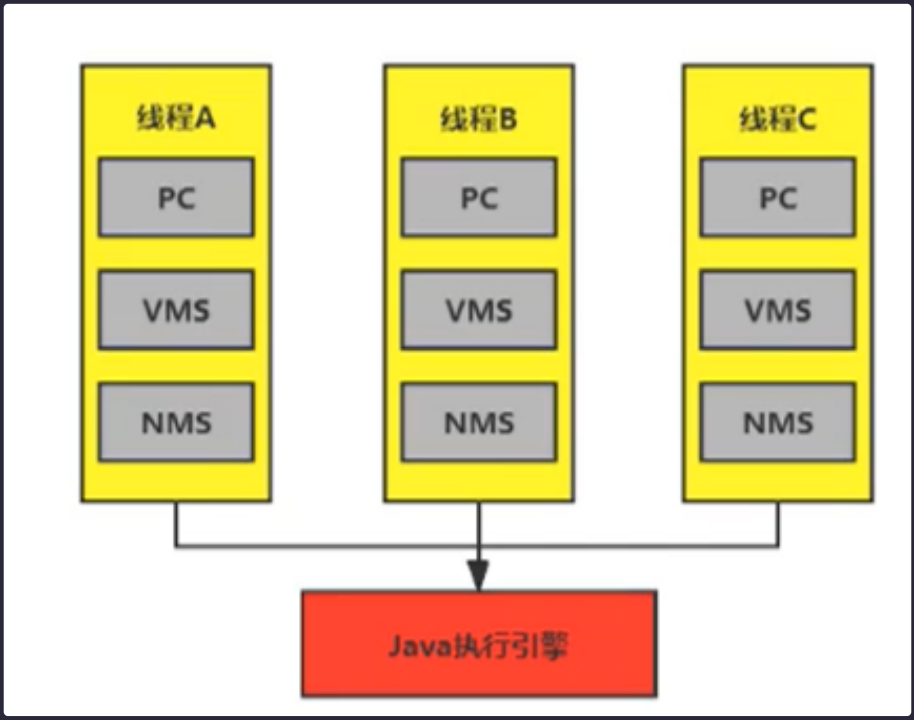
```
0: iconst_3
1: istore_1
2: iconst_4
3: istore_2
4: iload_1
5: iload_2
6: isub
7: ireturn
```



使用 PC 寄存器存储字节码指令地址有什么用呢？为什么使用 PC 寄存器记录当前线程的执行地址呢？

因为 CPU 需要不停的切换各个线程，这时候切换回来以后，就得知道接着从哪开始继续执行。

JVM 的字节码解释器就需要通过改变 PC 寄存器的值来明确下一条应该执行什么样的字节码指令。



PC 寄存器为什么被设定为私有的?

我们都知道所谓的多线程在一个特定的时间段内只会执行其中某一个线程的方法，CPU 会不停地做任务切换，这样必然导致经常中断或恢复，如何保证分毫无差呢？(为了能够准确地记录各个线程正在执行的当前字节码指令地址，最好的办法自然是为每一个线程都分配一个 PC 寄存器)，这样一来各个线程之间便可以进行独立计算，从而不会出现相互干扰的情况。

由于 CPU 时间片轮限制，众多线程在并发执行过程中，任何一个确定的时刻，一个处理器或者多核处理器中的一个内核，只会执行某个线程中的一条指令。

这样必然导致经常中断或恢复，如何保证分毫无差呢？每个线程在创建后，都会产生自己的程序计数器和栈帧，程序计数器在各个线程之间互不影响。

CPU 时间片

CPU 时间片即 CPU 分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片。

在宏观上：俄们可以同时打开多个应用程序，每个程序并行不悖，同时运行。

但在微观上：由于只有一个 CPU，一次只能处理程序要求的一部分，如何处理公平，一种方法就是引入时间片，每个程序轮流执行。