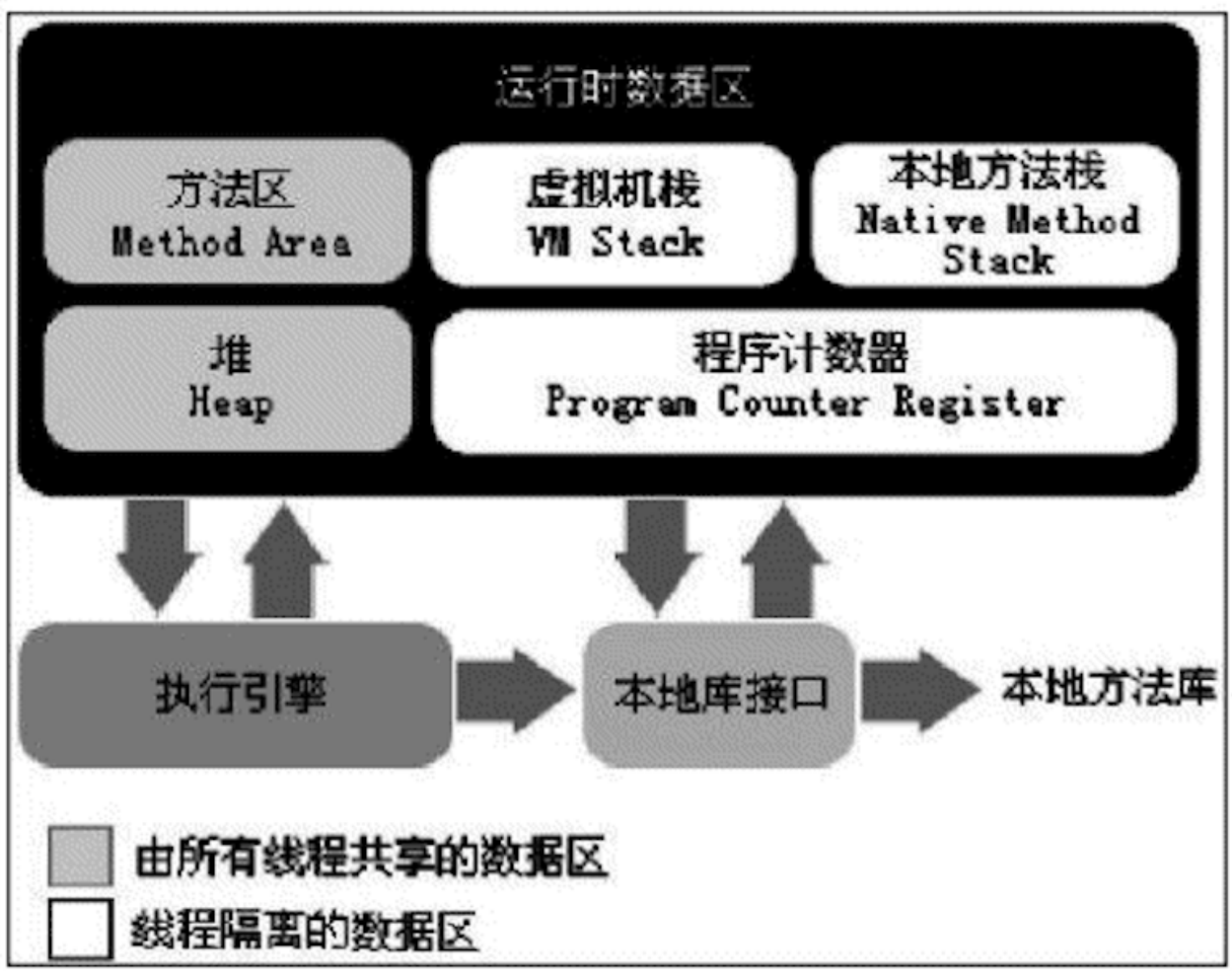
内存区域

15/7/26 15:55

1. 运行时数据区



2. 程序计数器（线程私有）

可以看做当前线程所执行字节码的行号指示器。
字节码解释器工作时通过改变这个计数器的值来取下一条要执行的指令，分支、循环、跳转、异常处理、线程恢复等都依赖计数器完成。
执行Native方法时，计数器值为Undefined。

3. Java虚拟机栈（线程私有）

生命周期与线程相同，描述了Java方法执行的内存模型：
每个方法在执行时会创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出口等信息。
每个方法从调用到执行完成的过程就是一个栈帧在虚拟机栈中入栈道出栈的过程。

局部变量表存放了编译期可知的各种基本数据类型、对象引用和returnAddress类型。所需的内存在编译期间完成分配，所以每个方法局部变量表所需空间是完全确定的，且运行期间不会变。

4. 本地方法栈（线程私有）

与java虚拟机栈类似，是为使用到的Native方法服务的。

5. Java堆（线程共享）

存放对象实例，还有数组。
Java堆可以处于物理上不连续的空间中，但逻辑上要连续
是垃圾回收的主要区域，基本采用分代收集算法。

6. 方法区（线程共享）

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等。（一般不会被回收）

7. 运行时常量池（方法区的一部分）

Class文件中除了有类的版本、字段、方法、接口等信息外还有常量池（用于存放编译期生成的各种字面量和符号引用）。常量池在类加载后进入方法区的运行时常量池。
运行期间也可以有常量产生进入运行时常量池。

8. 直接内存

不是运行时数据区的一部分，也不是java虚拟机规范中定义的内存区域。
在JDK1.4中新加入的New Input/Output类，引入了一种基于通道与缓冲区的I/O方式。可以使用Native函数库直接分配堆外内存，然后通过存储在Java堆中的DirectByteBuffer对象作为这亏内存的引用进行操作，免去在Java堆和Native堆中来回复制数据，显著提高性能。

9. 对象的创建过程

- 首先回去检查类是否加载过、解析和初始化过（常量池是否有类的符号引用），若没有，则先执行类加载过程。
- 类加载检查通过后，虚拟机为新生对象分配内存，将确定大小的内存从Java堆中划分出来。
- 虚拟机将分配到的内存空间都初始化为零值。
- 虚拟机对对象进行必要的设置，如是哪个类的实例、如何找到类的元数据信息、对象哈希码、对象GC分代年龄等。这些信息放在对象头中。
- 执行对象init方法

10. 从Java堆中划分内存的方法

- “指针碰撞”：假设Java堆中内存是绝对规则的，所有用过的在一边，空闲的在另一边，中间放着一个指针作为分界点，则分配内存时将指针想空闲空间那边移动相应大小的内存。
- “空闲列表”：已使用内存和空闲内存交错，虚拟机就必须维护一个列表，记录可用内存，分配时找一块足够大的内存空间给对象实例。
  *处理线程安全问题：一是使同步处理，二是预先给每个线程分配一小块内存，使分配动作在不同空间进行。

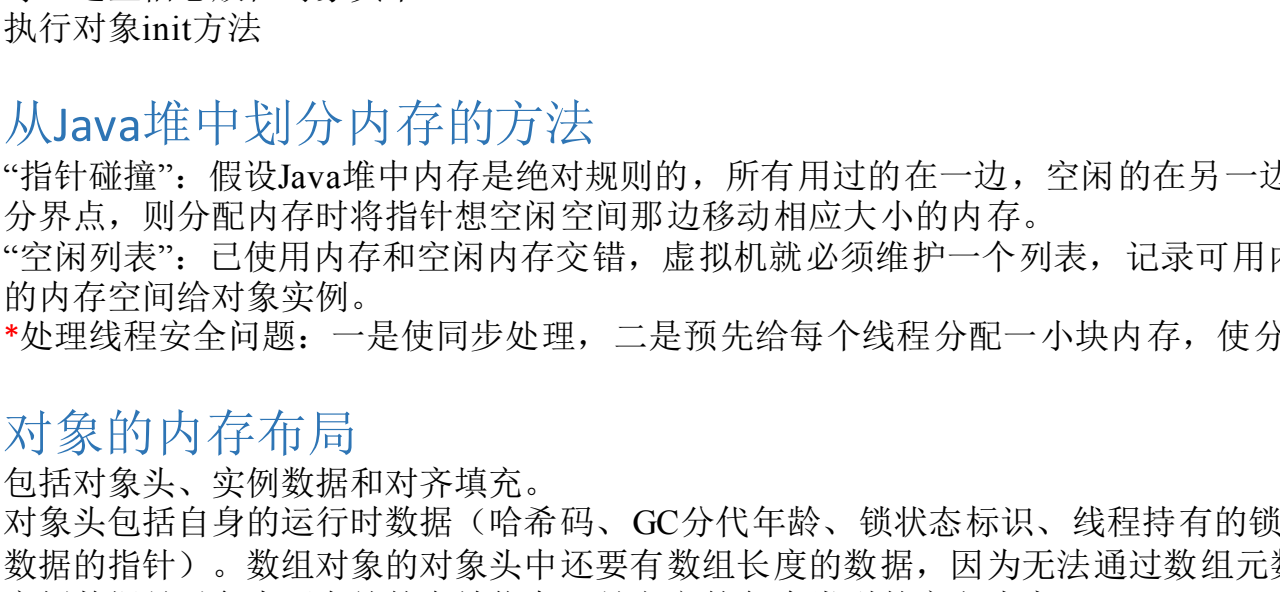
11. 对象的内存布局

包括对象头、实例数据和对齐填充。

- 对象头包括自身的运行时数据（哈希码、GC分代年龄、锁状态标识、线程持有的锁...）、类型指针（指向类元数据的指针）。数组对象的对象头中还要有数组长度的数据，因为无法通过数组元数据确定数组大小。
- 实例数据是对象真正存储的有效信息，是定义的各个类型的字段内容。
- 对齐填充不是必然存在的，只是用来对齐实例数据的。

12. 对象的定位访问

- 句柄访问：在java堆中划分出一块内存在作为句柄池。reference中存储对象的句柄地址，而句柄中包含了对象实例数据和类型数据各自的具体地址信息。
  优点：reference中的句柄地址稳定，在对象移动时（垃圾回收）只要改变句柄中的实例数据指针。



- 直接指针访问：reference中直接存储对象地址。
  优点：速度快，节省一次指针定位的时间开销。