1.方法区和堆是多个线程共享的；

java栈（JVM栈）、本地方法栈、程序计数器是各个线程独有的；

程序计数器：每一个线程都有一个，线程私有的，占用空间非常小，可以忽略不计；

用于存储指向下一条指令的内存地址，即将要执行的指令代码，由执行引擎读取下一条指令

QA:为什么使用PC寄存器来记录当前线程的执行地址？

**因为CPU需要不停的切换线程，等切换回来后，需要知道从哪继续开始执行；JVM的字节码解释器通过改变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。**

QA:PC寄存器为什么要线程私有？

**多线程在特定的时间内只会执行某一个线程，cpu会不断的切换任务，导致线程经常中断或恢复，为了能准确的记录各个线程正在执行的当前字节码的指令，最好的办法是为每一个线程分配一个PC寄存器**

2.方法区：方法区是线程共享的，所有字段和方法字节码及一下特殊方法，如构造函数、接口代码也在此定义，所有定义的方法的信息都存在此处，该区域属于共享区间。

静态变量（static）、常量（final）、类信息（Class）（构造函数、接口定义）、运行时的常量池在方法区，但是实例变量存在堆内存中，与方法区无关

栈：先进后出 ---好比一个桶 main()先执行，后结束

8大基本类型 + 对象引用 + 实例方法

栈内存，主管程序的运行，生命周期和线程同步，线程结束，栈内存释放，所以不存在垃圾回收问题。

队列：先进先出（FIFO first input first output）

堆：一个JVM只有一个堆，堆内存大小是可以调节的，存放new出来的对象

出现OOM情况：

1.一个启动类加载了大量的第三方jar包；

2.tomcat部署了太多的应用；

3.大量动态生成的反射类，被不断加载，使得堆内存溢出；

默认情况下，分配的总内存是电脑内存的1/4，初始化的内存是1/64；

-Xms:配置初始化内存 -Xmx:配置最大分配内存

//打印信息

-Xms1024m -Xmx1024m -XX:+printGCDetails

//输入dump，排查oom的原因

-Xms1024m -Xmx1024m -XX:+HeapDumpOnOutOfMemoryError

解决OOM：1.尝试扩大堆内存看结果;

2.分析内存，看哪个地方出的问题，使用工具（JProfiler）

新生区（伊甸园区、幸存0区、幸存1区）

老年区

永久区（元空间）

这个区域是常驻内存的，用来存放JDK自身携带的Class对象。interface元数据，存放的是Java运行时的一些环境或类信息，这个区域不存在垃圾回收，关闭虚拟机就会释放这个区域的内存。

jdk 1.6之前 永久代，常量池在方法区

jdk 1.7 永久代渐渐退化，去永久代，常量池在堆中

jdk 1.8以后 无永久代，常量池在元空间中(元空间逻辑上存在，物理上不存在)

GC(垃圾回收)-新生代、幸存区、老年代

1.JVM在进行GC时，并不是对这三个区域统一回收，大部分回收的是新生代；

轻GC（普通GC） -新生代

重GC -（全局GC）

2.每次GC，会将eden区中活着的对象放到幸存区，GC执行后，eden和幸存to区就会变成空的

3.当一个对象经历15（默认次数）次GC仍存活，就放到老年区中，可以通过-XX:MaxTenuringThreshold=15 这个属性来修改次数

**年轻代主要用复制算法（将幸存from区，复制到幸存to区，原来的from区变成to区），谁空谁是幸存to**

常用的算法：标记清除法、标记压缩法、复制算法、引用计数法

复制算法：

优点：没有内存碎片

缺点：浪费内存空间，多了一半空的区域（幸存to区）；极端情况下，对象100%存活

最佳使用场景：对象存活度低的情况， -- 新生区

标记清除法：1.扫描对象，标记活着的对象；

2.清除对象，把没有标记的对象清除

优点：不需要额外的空间

缺点：扫描两次，浪费时间，会产生内存碎片

标记压缩法：防止内存碎片，再次扫描，将存活的对象向一侧移动，增加了移动成本

内存效率（时间复杂度）：复制算法>标记清除法>标记压缩法

内存整齐度：复制算法=标记压缩法>标记清除

内存利用率：标记压缩法>标记清除法>复制算法

**没有最好的算法，只有最适合的算法 --分代收集算法**

年轻代：存活率低，复制算法；

老年代：区域大，存活率高，标记清除+标记压缩混合实现

native:

带有native关键字的说明java的作用范围访问不到了，要调用底层C语言的库；

进入本地方法栈，调用JNI（本地方法接口）；

JNI作用：扩展Java的使用，融合其他编程语言为java所用 C 、C++

背景：java诞生时，C和C++流行，如果想要立足就要有调它们的程序。因此，在内存中开辟了一个标记区域（本地方栈），来登记native方法。通过JNI来执行本地方法库。

使用：现在使用越来越少，除非通过java调用硬件（java驱动打印机、robot）

手写一个虚拟机 类加载器和执行引擎要考虑

类加载器加载过程：加载---链接（验证、准备、解析）---初始化

加载：1.通过类的全限定名获取类的二进制字节流文件

2.将字节流中的静态数据结构转化为方法区中运行时的数据结构

3.在内存中生成该类的.class文件，作为方法区这个类的各个数据访问入口

虚拟机自带的加载器：

1.启动类加载器（引导类加载器）嵌套到JVM里的，使用c、c++编写

没有父类加载器，不继承java.lang.classLoader

2.应用程序类加载器（系统类加载器）

程序默认的加载器

父类是扩展类加载器

双亲委派机制：

原理：如果一个类加载器收到了类加载请求，他自己先不加载，而是将请求委托给父类加载器加载；如果父类加载器还有父类就进一步向上委托，依次递归，直到请求到达顶层启动类加载器；如果父类加载器可以完成加载任务，就成功返回，如果父类无法完成加载，子类才会自己尝试去加载

优点：避免类的重复加载

保护程序安全，防止核心API会篡改（自定义类）