## JVM 面试题汇总

#### 1.什么是 JVM？它有什么作用？

答：JVM 是 Java Virtual Machine（Java 虚拟机）的缩写，顾名思义它是一个虚拟计算机，也是 Java 程序能够实现跨平台的基础。它的作用是加载 Java 程序，把字节码翻译成机器码再交由 CPU 执行的一个虚拟计算器。

#### 2.JVM 主要组成部分有哪些？

答：JVM 主要组成部分如下：

* 类加载器（ClassLoader）
* 运行时数据区（Runtime Data Area）
* 执行引擎（Execution Engine）
* 本地库接口（Native Interface）

#### 3.JVM 是如何工作的？

答：首先程序在执行之前先要把 Java 代码（.java）转换成字节码（.class），JVM 通过类加载器（ClassLoader）把字节码加载到内存中，但字节码文件是 JVM 的一套指令集规范，并不能直接交给底层操作系统去执行，因此需要特定的命令解析器执行引擎（Execution Engine） 将字节码翻译成底层机器码，再交由 CPU 去执行，CPU 执行的过程中需要调用本地库接口（Native Interface）来完成整个程序的运行。

#### 4.JVM 内存布局是怎样的？

答：不同虚拟机实现可能略微有所不同，但都会遵从 Java 虚拟机规范，Java 8 虚拟机规范规定，Java 虚拟机所管理的内存将会包括以下几个区域：

* 程序计数器（Program Counter Register）
* Java 虚拟机栈（Java Virtual Machine Stacks）
* 本地方法栈（Native Method Stack）
* Java 堆（Java Heap）
* 方法区（Methed Area）

**① 程序计数器**

程序计数器（Program Counter Register）是一块较小的内存空间，它可以看作是当前线程所执行的字节码的行号指示器。在虚拟机的概念模型里，字节码解析器的工作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，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。

由于 JVM 的多线程是通过线程轮流切换并分配处理器执行时间的方式来实现的，也就是任何时刻，一个处理器（或者说一个内核）都只会执行一条线程中的指令。因此为了线程切换后能恢复到正确的执行位置，每个线程都有独立的程序计数器。

如果线程正在执行 Java 中的方法，程序计数器记录的就是正在执行虚拟机字节码指令的地址，如果是 Native 方法，这个计数器就为空（undefined），因此该内存区域是唯一一个在 Java 虚拟机规范中没有规定 OutOfMemoryError 的区域。

**② Java 虚拟机栈**

Java 虚拟机栈（Java Virtual Machine Stacks）描述的是 Java 方法执行的内存模型，每个方法在执行的同时都会创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息，每个方法从调用直至执行完成的过程，都对应着一个线帧在虚拟机栈中入栈到出栈的过程。

* 如果线程请求的栈深度大于虚拟机所允许的栈深度就会抛出 StackOverflowError 异常。
* 如果虚拟机是可以动态扩展的，如果扩展时无法申请到足够的内存就会抛出 OutOfMemoryError 异常。

**③ 本地方法栈**

本地方法栈（Native Method Stack）与虚拟机栈的作用是一样的，只不过虚拟机栈是服务 Java 方法的，而本地方法栈是为虚拟机调用 Native 方法服务的。

在 Java 虚拟机规范中对于本地方法栈没有特殊的要求，虚拟机可以自由的实现它，因此在 Sun HotSpot 虚拟机直接把本地方法栈和虚拟机栈合二为一了。

**④ Java 堆**

Java 堆（Java Heap）是 JVM 中内存最大的一块，是被所有线程共享的，在虚拟机启动时候创建，Java 堆唯一的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，随着JIT编译器的发展和逃逸分析技术的逐渐成熟，栈上分配、标量替换优化的技术将会导致一些微妙的变化，所有的对象都分配在堆上渐渐变得不那么绝对了。

如果在堆中没有内存完成实例分配，并且堆不可以再扩展时，将会抛出 OutOfMemoryError。 Java 虚拟机规范规定，Java 堆可以处在物理上不连续的内存空间中，只要逻辑上连续即可，就像我们的磁盘空间一样。在实现上也可以是固定大小的，也可以是可扩展的，不过当前主流的虚拟机都是可扩展的，通过 -Xmx 和 -Xms 控制。

**⑤ 方法区**

方法区（Methed Area）用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。

很多人把方法区称作“永久代”（Permanent Generation），本质上两者并不等价，只是 HotSpot 虚拟机垃圾回收器团队把 GC 分代收集扩展到了方法区，或者说是用来永久代来实现方法区而已，这样能省去专门为方法区编写内存管理的代码，但是在 JDK 8 也移除了“永久代”，使用 Native Memory 来实现方法区。

当方法无法满足内存分配需求时会抛出 OutOfMemoryError 异常。

#### 5.在 Java 中负责字节码解释执行的是？

A：应用服务器  
B：垃圾回收器  
C：虚拟机  
D：编译器

答：C

#### 6.静态变量存储在哪个区？

A：栈区  
B：堆区  
C：全局区  
D：常量区

答：C

题目解析：栈区存放函数的参数值，局部变量的值等；堆区存放的是程序员创建的对象；全局区存放全局变量和静态变量；常量区存放常量字符串。

#### 7.垃圾回收算法有哪些？

答：垃圾回收算法如下。

* 引用计数器算法：引用计算器判断对象是否存活的算法是这样的，给每一个对象设置一个引用计数器，每当有一个地方引用这个对象的时候，计数器就加 1，与之相反，每当引用失效的时候就减 1。
* 可达性分析算法：在主流的语言的主流实现中，比如 Java、C#，甚至是古老的 Lisp 都是使用的可达性分析算法来判断对象是否存活的。这个算法的核心思路就是通过一些列的“GC Roots”对象作为起始点，从这些对象开始往下搜索，搜索所经过的路径称之为“引用链”。当一个对象到 GC Roots 没有任何引用链相连的时候，证明此对象是可以被回收的。
* 复制算法：复制算法是将内存分为大小相同的两块，当这一块使用完了，就把当前存活的对象复制到另一块，然后一次性清空当前区块。此算法的缺点是只能利用一半的内存空间。
* 标记-清除算法：此算法执行分两阶段，第一阶段从引用根节点开始标记所有被引用的对象，第二阶段遍历整个堆，把未标记的对象清除。此算法需要暂停整个应用，同时，会产生内存碎片。
* 标记-整理：此算法结合了“标记-清除”和“复制”两个算法的优点。分为两个阶段，第一阶段从根节点开始标记所有被引用对象，第二阶段遍历整个堆，把清除未标记对象并且把存活对象“压缩”到堆的其中一块，按顺序排放。此算法避免了“标记-清除”的碎片问题，同时也避免了“复制”算法的空间问题。

#### 8.哪些对象可以作为引用链的 Root 对象？

答：引用链的 Root 对象可以为以下内容：

* Java 虚拟机栈中的引用对象；
* 本地方法栈中 JNI（既一般说的 Native 方法）引用的对象；
* 方法区中类静态常量的引用对象；
* 方法区中常量的引用对象。

#### 9.对象引用关系都有哪些？

答：不管是引用计数法还是可达性分析算法都与对象的“引用”有关，这说明对象的引用决定了对象的生死，对象的引用关系如下。

* 强引用：在代码中普遍存在的，类似 Object obj = new Object() 这类引用，只要强引用还在，垃圾收集器永远不会回收掉被引用的对象。
* 软引用：是一种相对强引用弱化一些的引用，可以让对象豁免一些垃圾收集，只有当JVM 认为内存不足时，才会去试图回收软引用指向的对象，JVM 会确保在抛出 OutOfMemoryError 之前，清理软引用指向的对象。
* 弱引用：非必需对象，但它的强度比软引用更弱，被弱引用关联的对象只能生存到下一次垃圾收集发生之前。
* 虚引用：也称为幽灵引用或幻影引用，是最弱的一种引用关系，无法通过虚引用来获取一个对象实例，为对象设置虚引用的目的只有一个，就是当着个对象被收集器回收时收到一条系统通知。

#### 10.内存溢出和内存泄漏的区别是什么？

答：内存溢出和内存泄漏的区别如下：

* 内存溢出是指程序申请内存时，没有足够的内存，就会报错 OutOfMemory；
* 内存泄漏是指垃圾对象无法回收，可以使用 Memory Analyzer 等工具排出内存泄漏。

#### 11.垃圾回收的分类都有哪些？

答：垃圾回收的分类如下：

* 新生代回收器：Serial、ParNew、Parallel Scavenge
* 老年代回收器：Serial Old、Parallel Old、CMS
* 整堆回收器：G1

#### 12.分代垃圾回收器的组成部分有哪些？

答：分代垃圾回收器是由：新生代（Young Generation）和老生代（Tenured Generation）组成的，默认情况下新生代和老生代的内存比例是 1:2。

#### 13.新生代的组成部分有哪些？

答：新生代是由：Eden、Form Survivor、To Survivor 三个区域组成的，它们内存默认占比是 8:1:1。

#### 14.新生代垃圾回收是怎么执行的？

答：新生代垃圾回收的执行过程如下：

① Eden 区 + From Survivor 区存活着的对象复制到 To Survivor 区；  
② 清空 Eden 和 From Survivor 分区；  
③ From Survivor 和 To Survivor 分区交换（From 变 To，To 变 From）。

#### 15.为什么新生代有两个 Survivor 分区？

答：当新生代的 Survivor 分区为 2 个的时候，不论是空间利用率还是程序运行的效率都是最优的。

* 如果 Survivor 是 0 的话，也就是说新生代只有一个 Eden 分区，每次垃圾回收之后，存活的对象都会进入老生代，这样老生代的内存空间很快就被占满了，从而触发最耗时的 Full GC ，显然这样的收集器的效率是我们完全不能接受的。
* 如果 Survivor 分区是 1 个的话，假设把两个区域分为 1:1，那么任何时候都有一半的内存空间是闲置的，显然空间利用率太低不是最佳的方案。但如果设置内存空间的比例是 8:2 ，只是看起来似乎“很好”，假设新生代的内存为 100 MB（ Survivor 大小为 20 MB ），现在有 70 MB 对象进行垃圾回收之后，剩余活跃的对象为 15 MB 进入 Survivor 区，这个时候新生代可用的内存空间只剩了 5 MB，这样很快又要进行垃圾回收操作，显然这种垃圾回收器最大的问题就在于，需要频繁进行垃圾回收。
* 如果 Survivor 分区有 2 个分区，我们就可以把 Eden、From Survivor、To Survivor 分区内存比例设置为 8:1:1 ，那么任何时候新生代内存的利用率都 90% ，这样空间利用率基本是符合预期的。再者就是虚拟机的大部分对象都符合“朝生夕死”的特性，因此每次新对象的产生都在空间占比比较大的 Eden 区，垃圾回收之后再把存活的对象方法存入 Survivor 区，如果是 Survivor 区存活的对象，那么“年龄”就 +1 ，当年龄增长到 15 （可通过 -XX:+MaxTenuringThreshold 设定）对象就升级到老生代。

经过以上对比，可以得出结论，当新生代的 Survivor 分区为 2 个的时候，不论是空间利用率还是程序运行的效率都是最优的。

#### 16.什么是 CMS 垃圾回收器？

答：CMS（Concurrent Mark Sweep）一种以获得最短停顿时间为目标的收集器，非常适用 B/S 系统。

#### 17.CMS 垃圾回收器有哪些优缺点？

答：CMS 垃圾回收器的优点是使用多线程，标记清除垃圾的，它缺点如下。

* 对 CPU 资源要求敏感：CMS 回收器过分依赖于多线程环境，默认情况下，开启的线程数为（CPU 的数量 + 3）/ 4，当 CPU 数量少于 4 个时，CMS 对用户本身的操作的影响将会很大，因为要分出一半的运算能力去执行回收器线程；
* CMS 无法清除浮动垃圾：浮动垃圾指的是 CMS 清除垃圾的时候，还有用户线程产生新的垃圾，这部分未被标记的垃圾叫做“浮动垃圾”，只能在下次 GC 的时候进行清除；
* CMS 垃圾回收会产生大量空间碎片：CMS 使用的是标记-清除算法，所有在垃圾回收的时候回产生大量的空间碎片。

#### 18.什么是 G1 垃圾回收器？

答：G1 垃圾回收器是一种兼顾吞吐量和停顿时间的 GC 实现，是 JDK 9 以后的默认 GC 选项。G1 可以直观的设定停顿时间的目标，相比于 CMS CG，G1 未必能做到 CMS 在最好情况下的延时停顿，但是最差情况要好很多。

G1 GC 仍然存在着年代的概念，但是其内存结构并不是简单的条带式划分，而是类似棋盘的一个个 Region。Region 之间是复制算法，但整体上实际可看作是标记 - 整理（Mark-Compact）算法，可以有效地避免内存碎片，尤其是当 Java 堆非常大的时候，G1 的优势更加明显。

#### 19.垃圾回收的调优参数有哪些？

答：垃圾回收的常用调优如下：

* -Xmx:512 设置最大堆内存为 512 M；
* -Xms:215 初始堆内存为 215 M；
* -XX:MaxNewSize 设置最大年轻区内存；
* -XX:MaxTenuringThreshold=5 设置新生代对象经过 5 次 GC 晋升到老年代；
* -XX:PretrnureSizeThreshold 设置大对象的值，超过这个值的大对象直接进入老生代；
* -XX:NewRatio 设置分代垃圾回收器新生代和老生代内存占比；
* -XX:SurvivorRatio 设置新生代 Eden、Form Survivor、To Survivor 占比。

更多java学习资料如下：

[Java高级架构师课程(总共122门课程，1460GB)](https://www.consultdog.com/index.php/front/44/)

[最强java面试视频课程(21门课程，126GB)](https://www.consultdog.com/index.php/front/78/)