1. 内存模型以及分区，详细到每个区放什么？

方法区（永久区）、Java 堆、Java 栈（Java 虚拟机栈、Java 本地方法栈）、PC。

方法区：方法区有些虚拟机也叫做永久区，他是线程共有的，每个线程都能够去访问该区域。方法去一般存放一些类信息、常量（String 常量）、静态变量等等。。

Java 堆：首先 Java 堆也是线程共享的，线程之间能够共同访问同一片 Java 堆。Java 堆一般是垃圾回收的主要区域。这片区域主要存放一些对象实例和数组还有类对象的成员变量也都存放在 Java 堆中。

Java 堆一般还被分为两类区域，Eden 区域和 Survivor 区域，而 Survivor 区域又分 From Survivor 和 To Survivor 两种。他们大小的比例大概 8：1：1。在大部分情况下，对象都会在新生代 Eden 区域中分配内存，当 Eden 区域内存不够时就会触发 Micor GC，将 Eden 区域和其中一块 Survivor 区域中存活的对象放在另一块 Survivor 区域中。

Java 虚拟机栈：首先 Java 栈是线程私有的。执行每一个方法都会创建一个栈帧，在栈帧上会存放一些方法上面的局部变量、对象引用变量、方法出口等等的信息。

了解：程序计数器 PC：记录当前虚拟机正在执行的指令地址。如果执行 native 方法，则计数器置 null。

2. 堆里面的分区：Eden、From Survivor To Survivor，老年代的各自特点。

Java 堆一般会分成两类区域：新生代 和 老年代。

新生代又分为：Eden 区、Form Survivor 区、 To Survivor 区。比例 8：1：1

之所以这么划分，是为了方便垃圾回收。一般来说，我们创建的新对象都会存放在 Eden 分区上，直到 Eden 分区内存不够分配了就会触发 Minor GC，将 Eden 和其中一个 Survivor 分区中存活的对象复制到另一个 Survivor 当中。然后再把内存垃圾清空。

老年代区域中存放的一般是经过若干次垃圾回收依然存活下来的对象，这些对象就会转移到老年代当中。可以说老年区一般存放生命周期比较长的对象。

3. 对象的创建方法，对象的内存分配、对象的访问定位。

对象创建：语言层面上 通过 New 关键字来创建对象。

在虚拟机层面 检查指令的参数是否在常量池中有引用，并检查该引用类是否被加载、初始化过，如果没有才去加载这个类。

对象内存分配：在类完成加载后虚拟机就为该类对象分配内存。一般内存分配方式有两种：“指针碰撞” – 内存空间是整齐的、已用内存和空闲内存分开放。

“空闲列表” – 内存空间是已用内存和空闲内存交错排放。

对象访问定位：

“句柄访问“：通过在堆中开辟一部分空间来维护一个句柄池。通过栈内引用指向句柄池的句柄地址，再通过句柄地址间接的相应的对象实例。

优点：对象改变不用改动栈内存中的引用，只需改动句柄池指向具体对象实例

的指针即可。

“直接指针访问“：通过栈内引用直接指向堆中对象实例。

优点：节省了一次指针定位的时间开销，速度更快。

了解：对象的布局：对象头(Header)、实例数据(Instance Data)和对齐填充(Padding) - 8b^n

4. GC 的两种判定方法：引用计数法 和 可达性分析法

引用计数法 :通过一个引用计数器，当对象存在一个指向他的引用时，计数器 +1，当引用失效时，计数器 -1。计数器为 0 就代表对象没有指向他的引用，确认垃圾。

缺点：无法解决对象和对象之间嵌套引用的问题。对象 A 存在一个指向 B 的成员变量。而 B 也有指向 A 的成员变量。

可达性分析法：核心就是查看对象是否有到达 GC Roots 对象的一条引用链，不存在的话，就带别该对象没有指向他的引用，可以确认是垃圾。(2 次标记 –finalized())

GC Root 对象： 1. jvm 栈中的引用变量
2.方法区中的静态属性对象
3.类常量引用对象

5. GC 收集的三种方法：

1. 标记清除 --先标记可回收的对象（垃圾），然后将对象统一回收。
缺点：效率问题,标记和清除效率都不高；产生大量的不连续的内存碎片；
2. 标记整理 --标记存活对象 > 存活对象移向一端 > 回收边界之外的内存。
3. 复制算法 --标记存活对象 > 将存活对象移动到另一块分区 > 回收分区。

优化：分代回收 --新生代：存活率低，复制算法。(少量存活对象的复制)
--老年代：存活率高、没有其他空间担保，标记整理算法。

6. GC 收集器有哪些？CMS 收集器和 G1 收集器的特点。

CMS（标记 - 清除）：--（会产生大量连续的内存碎片）
初始标记 --标记 GCRoots 能够直接到达的对象，时间短
并发标记 --进行 GCRoots Trancing（可达性分析），时间长。
重新标记 --修正并发过程产生的变动。
并发清除 --回收内存，时间长。

G1（标记 - 整理）：
初始标记 --标记 GCRoots 能够直接到达的对象。时间短
并发标记 --并发进行 GCRoots 可达性分析。时间长
最终标记 --修正并发过程产生的变动。
筛选回收 --回收内存，时间长。

7. Minor GC 和 Full GC 分别发生时间/触发条件？

Minor GC 是指新生代（Eden、Survivor）的垃圾回收，执行的频率高，效率快。
Major GC 是指老年代的垃圾回收，不频繁。一般 MajorGC 由 MinorGC 触发。
Full GC 是指在清理整个堆的内存（包括新生代和老年代垃圾回收） ，速度慢。

发生时间：

Minor GC：当 Eden 内存空间不足时触发。

Major GC/Full GC：(1)Ststem.gc () 建议 JVM 进行 GC

(2)老年代空间不足

(3)方法去内存不足

(4) MinorGC 后进入老年代的平均大小大于可用内存

(5) 由 Eden 区、From Survivor 区向 To Survivor 区复制时，对象大小大于 To Survivor 区可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小

8. 几种常用内存调试工具：jstack、jmap、jhat、jstat..

用来跟踪堆栈的信息，内存的使用情况、去向…

防止内存溢出、内存泄露的发生…

-- 这方面没有做过多的深入了解，Android 上也有自己的内存检测工具（Studio Memory monitor、LeakCanary。

9. 类加载的五大过程：加载、验证、准备、解析、初始化

加载：通过类的全限类名获取字节流、然后转换成相应类信息、数据结构存入方法区。

验证：验证.Class 文件合法性。

准备：准备为类变量分配空间。

解析：解析类方法、类字段等信息。将常量池中的符号引用替换为直接引用。

初始化：初始化类和其他资源。

10. 双亲委派机制：

BootStrap ClassLoader	--引导类加载器、加载核心 API（jre/lib 包下的 Class）
Extension ClassLoader	--扩展类加载器、加载扩展包的类文件。
Application ClassLoader	--应用程序类加载器、加载 classpath 路径下的类文件。
Custom ClassLoader	--自定义类加载器、通过 ClassLoader.load()加载的类。

对于任意的类加载器，如果接收到类加载请求，都不会立即加载这个类。而是将这个请求委派给父类加载器去加载。每一层加载器都如此，直到到达顶层的引导类加载器。只有当父类加载器无法完成加载并返回时，子类加载器才尝试去加载该类。

11. 静态分派和动态分派

静态分派：通过静态类型定位方法执行版本的分派叫静态分派。

-- 静态分派典型应用是重载 Overload。

-- 静态分派发生在编译阶段。

动态分派：通过动态类型确认方法执行版本的分派叫动态分派。

--动态分派典型应用是重写 Override。

--动态分派发生在运行阶段。

注意 :静态类型指在编译阶段就知道的类型，动态类型指在运行时才能确认的类型。