JVM

JVM内存是如何划分的?

- 技术点：JVM内存管理
- 思路：分条解释每部分内存的作用
- 参考回答：JVM会用一段空间来存储执行程序期间需要用到的数据和相关信息，这段空间就是运行时数据区（Runtime Data Area），也就是常说的JVM内存。JVM会将它所管理的内存划分为线程私有数据区和线程共享数据区两大类：
  - 线程私有数据区包含：
    - 程序计数器：是当前线程所执行的字节码的行号指示器
    - 虚拟机栈：是Java方法执行的内存模型
    - 本地方法栈：是虚拟机使用到的Native方法服务
  - 线程共享数据区包含：
    - Java堆：用于存放几乎所有的对象实例和数组；是垃圾收集器管理的主要区域，也被称做“GC堆”；是Java虚拟机所管理的内存中最大的一块
    - 方法区：用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据；Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池（Constant Pool Table），用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放

Java中堆和栈的区别?

- 技术点：内存管理
- 思路：从存放数据和内存回收角度出发
- 参考回答： 在java中，堆和栈都是内存中存放数据的地方，具题区别是:
  - 栈内存：主要用来存放基本数据类型和局部变量；当在代码块定义一个变量时会在栈中为这个变量分配内存空间，当超过变量的作用域后这块空间就会被自动释放掉。
  - 堆内存：用来存放运行时创建的对象，比如通过new关键字创建出来的对象和数组；需要由Java虚拟机的自动垃圾回收器来管理。

谈谈垃圾回收机制？为什么引用计数器判定对象是否回收不可行？知道哪些垃圾回收算法?

- 技术点：垃圾回收机制
- 思路：从如何判定对象可回收、如果回收具体算法这两方面展开谈垃圾回收机制
- 参考回答：
- （1）判定对象可回收有两种方法：
  - 引用计数算法：给对象中添加一个引用计数器，每当有一个地方引用它时，计数器值就加1；当引用失效时，计数器值就减1；任何时刻计数器为0的对象就是不可能再被使用的。然而在主流的Java虚拟机里未选用引用计数算法来管理内存，主要原因是它难以解决对象之间相互循环引用的问题，

所以出现了另一种对象存活判定算法。

- 可达性分析法：通过一系列被称为『GC Roots』的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链，当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的。其中可作为GC Roots的对象：虚拟机栈中引用的对象，主要是指栈帧中的本地变量、本地方法栈中Native方法引用的对象、方法区中类静态属性引用的对象、方法区中常量引用的对象

- （2）回收算法有以下四种:
  - 分代收集算法：是当前商业虚拟机都采用的一种算法，根据对象存活周期的不同，将Java堆划分为新生代和老年代，并根据各个年代的特点采用最适当的收集算法。
    - 新生代：大批对象死去，只有少量存活。使用『复制算法』，只需复制少量存活对象即可。
      - 复制算法：把可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用尽后，把还存活着的对象『复制』到另外一块上面，再将这一块内存空间一次清理掉。

    - 老年代：对象存活率高。使用『标记－清理算法』或者『标记－整理算法』，只需标记较少的回收对象即可。
      - 标记-清除算法：首先『标记』出所有需要回收的对象，然后统一『清除』所有被标记的对象。
      - 标记-整理算法：首先『标记』出所有需要回收的对象，然后进行『整理』，使得存活的对象都向一端移动，最后直接清理掉端边界以外的内存。

Java中引用有几种类型？在Android中常用于什么情景?

- 技术点：Java引用类型
- 思路：分条解释每种类型的特点和适用场景
- 参考回答：引用的四种类型

  - 强引用（StrongReference）：具有强引用的对象不会被GC；即便内存空间不足，JVM宁愿抛出OutOfMemoryError使程序异常终止，也不会随意回收具有强引用的对象。

  - 软引用（SoftReference）：只具有软引用的对象，会在内存空间不足的时候被GC；软引用常用来实现内存敏感的高速缓存。

  - 弱引用（WeakReference）：只被弱引用关联的对象，无论当前内存是否足够都会被GC；强度比软引用更弱，常用于描述非必需对象；常用于解决内存泄漏的问题

  - 虚引用（PhantomReference）：仅持有虚引用的对象，在任何时候都可能被GC；常用于跟踪对象被GC回收的活动；必须和引用队列 （ReferenceQueue）联合使用，当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。

类加载的全过程是怎样的？什么是双亲委派模型?

- 技术点：类加载机制、双亲委派模型
- 思路：类加载机制的含义以及每个阶段的作用，在解释双亲委派模型之前需要先理解类加载器，

- 参考回答：

- （1）类加载机制：是虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可被虚拟机直接使用的Java类型的过程。另外，类型的加载、连接和初始化过程都是在程序运行期完成的，从而通过牺牲一些性能开销来换取Java程序的高度灵活性。下面介绍类加载每个阶段的任务：

    - 加载（Loading）：通过类的全限定名来获取定义此类的二进制字节流；将该二进制字节流所代表的静态存储结构转化为方法区的运行时数据结构，该数据存储数据结构由虚拟机实现自行定义；在内存中生成一个代表这个类的java.lang.Class对象，它将作为程序访问方法区中的这些类型数据的外部接口

    - 验证（Verification）：确保Class文件的字节流中包含的信息符合当前虚拟机的要求，包括文件格式验证、元数据验证、字节码验证和符号引用验证

    - 准备（Preparation）：为类变量分配内存，因为这里的变量是由方法区分配内存的，所以仅包括类变量而不包括实例变量，后者将会在对象实例化时随着对象一起分配在Java堆中；设置类变量初始值，通常情况下零值

    - 解析（Resolution）：虚拟机将常量池内的符号引用替换为直接引用的过程

    - 初始化（Initialization）：是类加载过程的最后一步，会开始真正执行类中定义的Java字节码。而之前的类加载过程中，除了在『加载』阶段用户应用程序可通过自定义类加载器参与之外，其余阶段均由虚拟机主导和控制

- （2）类加载器：不仅用于加载类，还和这个类本身一起作为在JVM中的唯一标识。常见类加载器类型有：

    - 启动类加载器：是虚拟机自身的一部分
    - 扩展类加载器、应用程序类加载器、自定义类加载器：独立于虚拟机外部

- （3）双亲委派模型：表示类加载器之间的层次关系。

    - 前提：除了顶层启动类加载器外，其余类加载器都应当有自己的父类加载器，且它们之间关系一般不会以继承（Inheritance）关系来实现，而是通过组合（Composition）关系来复用父加载器的代码。

    - 工作过程：若一个类加载器收到了类加载的请求，它先会把这个请求委派给父类加载器，并向上传递，最终请求都传送到顶层的启动类加载器中。只有当父加载器反馈自己无法完成这个加载请求时，子加载器才会尝试自己去加载。

工作内存和主内存的关系？在Java内存模型有哪些可以保证并发过程的原子性、可见性和有序性的措施？

- 技术点：JVM内存模型、线程安全
- 思路：理解Java内存模型的结构
- 参考回答：Java内存模型就是通过定义程序中各个变量的访问规则，即在虚拟机中将变量存储到内存和

从内存中取出变量这样的底层细节。

- 其中，主内存（Main Memory）是所有变量的存储位置，每条线程还有自己的工作内存，用于保存被该线程使用到的变量的主内存副本拷贝。为了获取更好的运行速度，虚拟机可能会让工作内存优先存储于寄存器和高速缓存中。
- 保证并发过程的原子性、可见性和有序性的措施：
  - 原子性（Atomicity）：一个操作要么都执行要么都不执行。
    - 可直接保证的原子性变量操作有：read、load、assign、use、store和write，因此可认为基本数据类型的访问读写是具备原子性的。
    - 若需要保证更大范围的原子性，可通过更高层次的字节码指令monitorenter和monitorexit来隐式地使用lock和unlock这两个操作，反映到Java代码中就是同步代码块synchronized关键字。

  - 可见性（Visibility）：当一个线程修改了共享变量的值，其他线程能够立即得知这个修改。
    - 通过在变量修改后将新值同步回主内存，在变量读取前从主内存刷新变量值这种依赖主内存作为传递媒介的方式来实现。
    - 提供三个关键字保证可见性：volatile能保证新值能立即同步到主内存，且每次使用前立即从主内存刷新；synchronized对一个变量执行unlock操作之前可以先把此变量同步回主内存中；被final修饰的字段在构造器中一旦初始化完成且构造器没有把this的引用传递出去，就可以在其他线程中就能看见final字段的值。

  - 有序性（Ordering）：程序代码按照指令顺序执行。
    - 如果在本线程内观察，所有的操作都是有序的，指“线程内表现为串行的语义”；如果在一个线程中观察另一个线程，所有的操作都是无序的，指“指令重排序”现象和“工作内存与主内存同步延迟”现象。
    - 提供两个关键字保证有序性：volatile 本身就包含了禁止指令重排序的语义；synchronized保证一个变量在同一个时刻只允许一条线程对其进行lock操作，使得持有同一个锁的两个同步块只能串行地进入。