1.JVM 的主要组成部分？及其作用？

- 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）

组件的作用： 首先通过类加载器（ClassLoader）会把 Java 代码转换成字节码，运行时数据区（Runtime Data Area）再把字节码加载到内存中，而字节码文件只是 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系统指令，再交由 CPU 去执行，而这个过程中需要调用其他语言的本地库接口（Native Interface）来实现整个程序的功能。

2. 说一下 JVM 运行时数据区？

不同虚拟机的运行时数据区可能略微有所不同，但都会遵从 Java 虚拟机规范， Java 虚拟机规范规定的区域分为以下 5 个部分：

- 程序计数器（Program Counter Register）：当前线程所执行的字节码的行号指示器，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；
- Java 虚拟机栈（Java Virtual Machine Stacks）：用于存储局部变量表、操作数栈、动态链接、方法出口等信息；
- 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；
- Java 堆（Java Heap）：Java 虚拟机中内存最大的一块，是被所有线程共享的，几乎所有的对象实例都在这里分配内存；
- 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。

3.Java 内存分配

• 寄存器：我们无法控制。

• 静态域：static 定义的静态成员。

• 常量池：编译时被确定并保存在 .class 文件中的（final）常量值和一些文本修饰的符号引用（类和接口的全限定名，字段的名称和描述符，方法和名称和描述符）。

• 非 RAM 存储：硬盘等永久存储空间。

• 堆内存：new 创建的对象和数组，由 Java 虚拟机自动垃圾回收器管理,存取速度慢。

• 栈内存：基本类型的变量和对象的引用变量（堆内存空间的访问地址），速度快，可以共享，但是大小与生存期必须确定，缺乏灵活性。

4. 说一下 JVM 调优的工具？

JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是 jconsole 和 jvisualvm 这两款视图监控工具。

• jconsole：用于对 JVM 中的内存、线程和类等进行监控；

• jvisualvm：JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等。

5. 常用的 JVM 调优的参数都有哪些？

• -Xms2g：初始化推大小为 2g；

• -Xmx2g：堆最大内存为 2g；

- -XX:NewRatio=4：设置年轻的和老年代的内存比例为 1:4；
- -XX:SurvivorRatio=8：设置新生代 Eden 和 Survivor 比例为 8:2；
- -XX:+UseParNewGC：指定使用 ParNew + Serial Old 垃圾回收器组合；
- -XX:+UseParallelOldGC：指定使用 ParNew + ParNew Old 垃圾回收器组合；
- -XX:+UseConcMarkSweepGC：指定使用 CMS + Serial Old 垃圾回收器组合；
- -XX:+PrintGC：开启打印 gc 信息；
- -XX:+PrintGCDetails：打印 gc 详细信息。

6.Java 堆的结构是什么样子的？什么是堆中的永久代（Perm Gen space）？

JVM 的堆是运行时数据区，所有类的实例和数组都是在堆上分配内存。它在 JVM 启动的时候被创建。对象所占的堆内存是由自动内存管理系统也就是垃圾收集器回收。

堆内存是由存活和死亡的对象组成的。存活的对象是应用可以访问的，不会被垃圾回收。死亡的对象是应用不可访问尚且还没有被垃圾收集器回收掉的对象。一直到垃圾收集器把这些 对象回收掉之前，他们会一直占据堆内存空间。

7.什么是类加载器，类加载器有哪些？

实现通过类的权限定名获取该类的二进制字节流的代码块叫做类加载器。

主要有一下四种类加载器：

- 启动类加载器（Bootstrap ClassLoader）用来加载 Java 核心类库，无法被 Java 程序直接引用。

• 扩展类加载器（extensions class loader）：它用来加载 Java 的扩展库。Java 虚拟机的实现会提供一个扩展库目录。该类加载器在此目录里面查找并加载 Java 类。

• 系统类加载器（system class loader）：它根据 Java 应用的类路径（CLASSPATH）来加载 Java 类。一般来说，Java 应用的类都是由它来完成加载的。可以通过 ClassLoader.getSystemClassLoader() 来获取它。

• 用户自定义类加载器，通过继承 java.lang.ClassLoader 类的方式实现。

8.类加载器双亲委派模型机制？

当一个类收到了类加载请求时，不会自己先去加载这个类，而是将其委派给父类，由父类去加载，如果此时父类不能加载，反馈给子类，由子类去完成类的加载。

9.Java 类加载过程？

Java 类加载需要经历一下 7 个过程：

1. 加载

加载是类加载的第一个过程，在这个阶段，将完成一下三件事情：

• 通过一个类的全限定名获取该类的二进制流。

• 将该二进制流中的静态存储结构转化为方法去运行时数据结构。

• 在内存中生成该类的 Class 对象，作为该类的数据访问入口。

2. 验证

验证的目的是为了确保 Class 文件的字节流中的信息不回危害到虚拟机.在该阶段主要完成以下四钟验证:

• 文件格式验证：验证字节流是否符合 Class 文件的规范，如主次版本号是否在当前虚拟机范围内，常量池中的常量是否有不被支持的类型.

• 元数据验证:对字节码描述的信息进行语义分析，如这个类是否有父类，是否集成了不被继承的类等。

• 字节码验证：是整个验证过程中最复杂的一个阶段，通过验证数据流和控制流的分析，确定程序语义是否正确，主要针对方法体的验证。如：方法中的类型转换是否正确，跳转指令是否正确等。

• 符号引用验证：这个动作在后面的解析过程中发生，主要是为了确保解析动作能正确执行。

3. 准备

准备阶段是为类的静态变量分配内存并将其初始化为默认值，这些内存都将在方法区中进行分配。准备阶段不分配类中的实例变量的内存，实例变量将会在对象实例化时随着对象一起分配在 Java 堆中。

public static int value=123;//在准备阶段 value 初始值为 0 。在初始化阶段才会变为 123 。

4. 解析

该阶段主要完成符号引用到直接引用的转换动作。解析动作并不一定在初始化动作完成之前，也有可能在初始化之后。

5. 初始化

初始化时类加载的最后一步，前面的类加载过程，除了在加载阶段用户应用程序可以通过自定义类加载器参与之外，其余动作完全由

虚拟机主导和控制。到了初始化阶段，才真正开始执行类中定义的 Java 程序代码。

6. 使用
7. 卸载

10.GC 是什么？为什么要有 GC？

GC 是垃圾收集的意思（GabageCollection），内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。

11.简述 Java 垃圾回收机制

在 Java 中，程序员是不需要显示的去释放一个对象的内存的，而是由虚拟机自行执行。在 JVM 中，有一个垃圾回收线程，它是低优先级的，在正常情况下是不会执行的，只有在虚拟机空闲或者当前堆内存不足时，才会触发执行，扫面那些没有被任何引用的对象，并将它们添加到要回收的集合中，进行回收。

12.垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆（heap）中的所有对象。通过这种方式确定哪些对象是" 可达的"，哪些对象是"不可达的"。当 GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。可以。程序员可以手

动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

13.Java 中会存在内存泄漏吗，请简单描述

所谓内存泄露就是指一个不再被程序使用的对象或变量一直被占据在内存中。Java 中有垃圾回收机制，它可以保证一对象不再被引用的时候，即对象变成了孤儿的时候，对象将自动被垃圾回收器 从内存中清除掉。

由于 Java 使用有向图的方式进行垃圾回收管理， 可以消除引用循环的问题，例如有两个对象，相互引用，只要它们 和根进程不可达的，那么 GC 也是可以回收它们的，例如下面的代码可以看到这种情况的内存回收：

```
import java.io.IOException;

public class GarbageTest {

    /**

    *    @param args

    *    @throws IOException

    */

    public static void main(String[] args) throws IOException
    {

        // TODO Auto-generated method stub

        try {

            gcTest();

        } catch (IOException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        System.out.println("has exited gcTest!"); System.in.read();
        System.in.read(); System.out.println("out begin gc!");
        for(int i=0;i<100;i++)
        {
            System.gc(); System.in.read(); System.in.read();
        }

    }
```

```
    private static void gcTest() throws IOException { System.in.read();
    System.in.read();

    Person p1 = new Person(); System.in.read(); System.in.read();
    Person p2 = new Person(); p1.setMate(p2); p2.setMate(p1);
    System.out.println("before exit gctest!"); System.in.read();

    System.in.read(); System.gc();
    System.out.println("exit gctest!");

}

private static class Person
{

    byte[] data = new byte[20000000]; Person mate = null;
    public void setMate(Person other)
    {
        mate = other;
    }

}
```

14.深拷贝和浅拷贝

简单来讲就是复制、克隆。

Person p=new Person(“张三”);

浅拷贝就是对对象中的数据成员进行简单赋值，如果存在动态成员或者指针就会报错。 深拷贝就是对对象中存在的动态成员或指针重新开辟内存空间。

15.System.gc() 和 Runtime.gc() 会做什么事情？

这两个方法用来提示 JVM 要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于 JVM 的。

16.finalize() 方法什么时候被调用？析构函数 (finalization) 的目的是什么？

垃圾回收器（garbage colector）决定回收某对象时，就会运行该对象的 finalize() 方法但是在 Java 中很不幸，如果内存总是充足的，那么垃圾回收可能永远不会进行，也就是说 filalize() 可能永远不被执行，显然指望它做收尾工作是靠不住的。那么 finalize() 究竟是做什么的呢？ 它最主要的用途是回收特殊渠道申请的内存。Java 程序有垃圾回收器，所以一般情况下内存问题不用程序员操心。但有一种 JNI（Java Native Interface）调用 non-Java 程序（C 或 C++）， finalize() 的工作就是回收这部分的内存。

17.如果对象的引用被置为 null，垃圾收集器是否会立即释放对象占用的内存？

不会，在下一个垃圾回收周期中，这个对象将是可被回收的。

18.什么是分布式垃圾回收（DGC）？它是如何工作的？

DGC 叫做分布式垃圾回收。RMI 使用 DGC 来做自动垃圾回收。因为 RMI 包含了跨虚拟机的远程对象的引用，垃圾回收是很困难的。DGC 使用引用计数算法来给远程对象提供自动内存管理。

19.串行（serial）收集器和吞吐量（throughput）收集器的区别是什么？

吞吐量收集器使用并行版本的新生代垃圾收集器，它用于中等规模和大规模数据的应用程序。 而串行收集器对大多数的小应用（在现代处理器上需要大概 100M 左右的内存）就足够了。

20.在 Java 中，对象什么时候可以被垃圾回收？

当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。

21.简述 Java 内存分配与回收速率以及 Minor GC 和 Major GC。

- 对象优先在堆的 Eden 区分配
- 大对象直接进入老年代
- 长期存活的对象将直接进入老年代

当 Eden 区没有足够的空间进行分配时，虚拟机会执行一次 Minor GC。Minor GC 通常发生在新生代的 Eden 区，在这个区的对象生存期短，往往发生 Gc 的频率较高，回收速度比较快；Full GC/Major GC 发生在老年代，一般情况下，触发老年代 GC 的时候不会触发 Minor GC，但是通过配置，可以在 Full GC 之前进行一次 Minor GC 这样可以加快老年代的回收速度。

22.JVM 的永久代中会发生垃圾回收么？

垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值， 会触发完全垃圾回收（Full GC）。

注：Java 8 中已经移除了永久代，新加了一个叫做元数据区的 native 内存区。

23.Java 中垃圾收集的方法有哪些？

标记 - 清除：这是垃圾收集算法中最基础的，根据名字就可以知道，它的思想就是标记哪些要被回收的对象，然后统一回收。这种方法很简单，但是会有两个主要问题：

1. 效率不高，标记和清除的效率都很低；
2. 会产生大量不连续的内存碎片，导致以后程序在分配较大的 对象时，由于没有充足的连续内存而提前触发一次 GC 动作。

复制算法：为了解决效率问题，复制算法将可用内存按容量划分为相等的两部分，然后每次只使用其中的一块，当一块内存用完时，

就将还存活的对象复制到第二块内存上，然后一次性清楚完第一块内存，再将第二块上的对象复制到第一块。但是这种方式，内存的代价太高，每次基本上都要浪费一般的内存。

于是将该算法进行了改进，内存区域不再是按照 1:1 去划分，而 是将内存划分为 8:1:1 三部分，较大那份内存交 Eden 区，其余是两块较小的内存区叫 Survior 区。每次都会优先使用 Eden 区， 若 Eden 区满，就将对象复制到第二块内存区上，然后清除 Eden 区，如果此时存活的对象太多，以至于 Survivor 不够时，会将这些对象通过分配担保机制复制到老年代中。

（java 堆又分为新生代和老年代） 标记 - 整理：该算法主要是为了解决标记 - 清除，产生大量内存碎片的问题；当对象存活率较高时，也解决了复制算法的效率问题。它的不同之处就是在清除对象的时候现将可回收对象移动到一端， 然后清除掉端边界以外的对象，这样就不会产生内存碎片了。

分代收集：现在的虚拟机垃圾收集大多采用这种方式，它根据对象的生存周期，将堆分为新生代和老年代。在新生代中，由于对象生存期短，每次回收都会有大量对象死去，那么这时就采用复制算法。老年代里的对象存活率较高，没有额外的空间进行分配担保。

24. 堆栈的区别？

- 功能方面：堆是用来存放对象的，栈是用来执行程序的。
- 共享性：堆是线程共享的，栈是线程私有的。
- 空间大小：堆大小远远大于栈。

25. 队列和栈是什么？有什么区别？

队列和栈都是被用来预存储数据的。

队列允许先进先出检索元素，但也有例外的情况，Deque 接口允许从两端检索元素。

栈和队列很相似，但它运行对元素进行后进先出进行检索。

26.什么是双亲委派模型？

在介绍双亲委派模型之前先说下类加载器。对于任意一个类，都需要由加载它的类加载器和这个类本身一同确立在 JVM 中的唯一性，每一个类加载器，都有一个独立的类名称空间。类加载器就是根据指定全限定名称将 class 文件加载到 JVM 内存，然后再转化为 class 对象。

类加载器分类：

• 启动类加载器（Bootstrap ClassLoader），是虚拟机自身的一部分，用来加载 Java_HOME/lib/目录中的，或者被 -Xbootclasspath 参数所指定的路径中并且被虚拟机识别的类库；

• 其他类加载器：

• 扩展类加载器（Extension ClassLoader）：负责加载<java_home style="box-sizing: border-box; outline: 0px !important;">\lib\ext 目录或 Java. ext. dirs 系统变量指定的路径中的所有类库；

• 应用程序类加载器（Application ClassLoader）。负责加载用户类路径（classpath）上的指定类库，我们可以直接使用这个类加载器。一般情况，如果我们没有自定义类加载器默认就是用这个加载器。

双亲委派模型：如果一个类加载器收到了类加载的请求，它首先不会自己去加载这个类，而是把这个请求委派给父类加载器去完成，每一层的类加载器都是如此，这样所有的加载请求都会被传送到顶层的启动类加载器中，只有当父加载无法完成加载请求（它的搜索范围中没找到所需的类）时，子加载器才会尝试去加载类。

27.说一下类装载的执行过程？

类装载分为以下 5 个步骤：

- 加载：根据查找路径找到相应的 class 文件然后导入；
- 检查：检查加载的 class 文件的正确性；
- 准备：给类中的静态变量分配内存空间；
- 解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址；
- 初始化：对静态变量和静态代码块执行初始化工作。

28. 怎么判断对象是否可以被回收？

一般有两种方法来判断：

- 引用计数器：为每个对象创建一个引用计数，有对象引用时计数器 +1，引用被释放时计数 -1，当计数器为 0 时就可以被回收。它有一个缺点不能解决循环引用的问题；
- 可达性分析：从 GC Roots 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，则证明此对象是可以被回收的。

29.Java 中都有哪些引用类型？

- 强引用：发生 gc 的时候不会被回收。
- 软引用：有用但不是必须的对象，在发生内存溢出之前会被回收。
- 弱引用：有用但不是必须的对象，在下一次 GC 时会被回收。
- 虚引用（幽灵引用/幻影引用）：无法通过虚引用获得对象，用 PhantomReference 实现虚引用，虚引用的用途是在 gc 时返回一个通知。

30.说一下 JVM 有哪些垃圾回收算法？

• 标记-清除算法：标记无用对象，然后进行清除回收。缺点：效率不高，无法清除垃圾碎片。

• 标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存。

• 复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。缺点：内存使用率不高，只有原来的一半。

• 分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法。

31.说一下 JVM 有哪些垃圾回收器？

• Serial：最早的单线程串行垃圾回收器。

• Serial Old：Serial 垃圾回收器的老年版本，同样也是单线程的，可以作为 CMS 垃圾回收器的备选预案。

• ParNew：是 Serial 的多线程版本。

• Parallel 和 ParNew 收集器类似是多线程的，但 Parallel 是吞吐量优先的收集器，可以牺牲等待时间换取系统的吞吐量。

• Parallel Old 是 Parallel 老生代版本，Parallel 使用的是复制的内存回收算法，Parallel Old 使用的是标记-整理的内存回收算法。

• CMS：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。

• G1：一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。

32. 详细介绍一下 CMS 垃圾回收器？

CMS 是英文 Concurrent Mark-Sweep 的简称，是以牺牲吞吐量为代价来获得最短回收停顿时间的垃圾回收器。对于要求服务器响应速度的应用上，这种垃圾回收器非常适合。在启动 JVM 的参数加上"-XX:+UseConcMarkSweepGC"来指定使用 CMS 垃圾回收器。

CMS 使用的是标记-清除的算法实现的，所以在 gc 的时候回产生大量的内存碎片，当剩余内存不能满足程序运行要求时，系统将会出现 Concurrent Mode Failure，临时 CMS 会采用 Serial Old 回收器进行垃圾清除，此时的性能将会被降低。

33. 新生代垃圾回收器和老生代垃圾回收器都有哪些？有什么区别？

- 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

新生代垃圾回收器一般采用的是复制算法，复制算法的优点是效率高，缺点是内存利用率低；老年代回收器一般采用的是标记-整理的算法进行垃圾回收。

34. 简述分代垃圾回收器是怎么工作的？

分代回收器有两个分区：老生代和新生代，新生代默认的空间占比总空间的 1/3，老生代的默认占比是 2/3。

新生代使用的是复制算法，新生代里有 3 个分区：Eden、To Survivor、From Survivor，它们的默认占比是 8:1:1，它的执行流程如下：

- 把 Eden + From Survivor 存活的对象放入 To Survivor 区；
- 清空 Eden 和 From Survivor 分区；
- From Survivor 和 To Survivor 分区交换，From Survivor 变 To Survivor，To Survivor 变 From Survivor。

每次在 From Survivor 到 To Survivor 移动时都存活的对象，年龄就 +1，当年龄到达 15（默认配置是 15）时，升级为老生代。大对象也会直接进入老生代。

老生代当空间占用到达某个值之后就会触发全局垃圾收回，一般使用标记整理的执行算法。以上这些循环往复就构成了整个分代垃圾回收的整体执行流程。