JVM

1. Java 内存区域（运行时数据区）（重点）

（1）程序计数器：线程私有。记录当前线程执行的位置，**保存 JVM 中下一条所要执行的字节码指令的地址**！

（2）java 虚拟机栈：线程私有，生命周期与线程相同，每个方法在执行的同时都会创建一个栈帧用于存储局部变量表、操作数栈、动态链接、方法返回地址等信息。每一个方法从被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。（栈内存用来存储局部变量和方法调用。）

（3）本地方法栈：线程私有，本地方法栈与虚拟机栈所发挥的作用是非常相似的，它们之间的区别不过是虚拟机栈为虚拟机执行 Java 方法（也就是字节码）服务，而本地方法栈则为虚拟机使用到的 *Native* 方法服务。

（4）堆：线程共享，java 内存最大的一块，所有对象的实例和数组都存放在堆中，是垃圾收集器管理的主要区域

（5）方法区：线程共享，存放已被虚拟机加载的信息、常量、静态变量，编译后的代码等数据。

方法区的**垃圾收集**主要回收两部分内容：废弃的常量和不再使用的类型。

2. java 堆相关

由于 java 堆是垃圾收集器管理的主要区域，因此也被称作 **GC 堆（Garbage Collected Heap）**。从垃圾回收的角度，由于现在收集器基本都采用分代垃圾收集算法，所以 Java 堆还可以细分为：新生代和老年代；再细致一点有：Eden、Survivor、Old 等空间。进一步划分的目的是更好地回收内存，或者更快地分配内存。

新生代：

大部分情况，对象都会首先在 Eden 区域分配，在一次新生代垃圾回收后，如果对象还存活，eden 区域的对象则会进入 survivor 区，并且对象的年龄会加 1，直到 15 岁后进入老年代。

空间担保：

当创建的对象大于 eden，先进行一次新生代垃圾回收，如果还是无法创建新的对象，会通过空间担保策略提前进入进入老年代。当 Surivivor 区满了的时候，对象也会分配到老年代。

动态年龄：

什么是动态年龄勒，这是堆中的另一个，担保策略了，它会去判断我们 Surivivor 的区中，相同年龄的对象大于 Surivivor 区一半的时候，那么他就会判定此时这些对象已经能够很好的存活了，所以他们就集体被丢到老年代了。

老年代：

老年代存放的都是一些存活时间较长的对象，很少产生垃圾回收，一旦产生，耗时是新生代垃圾回收的 10 倍，而我们老年代快要存满时进入了一个对象，这时会产生一次老年代垃圾回收，如果 GC 结束后，还是无法存放对象的话此时就会报 OOM 异常。

3. 堆与栈的区别

最主要的区别就是栈内存用来存储局部变量和方法调用。

而堆内存用来存储 Java 中的对象。无论是成员变量，局部变量，还是类变量，它们指向的对象都存储在堆内存中。

4. 虚拟机栈内存溢出的情况

（1）虚拟机栈中，**栈帧过多**（方法无限递归）导致栈内存溢出，这种情况比较**常见。**当线程请求栈的深度超过当前 Java 虚拟机栈的最大深度的时候，就抛出 StackOverFlowError 错误；

（2）每个栈帧**所占用内存过大**(某个/某几个栈帧内存直接超过虚拟机栈最大内存)，这种情况比较**少见**!

5. GC（Garbage Collected）的两种判定方法（如何判断一个对象是否存活）

（1）引用计数法：指的是如果某个地方引用了这个对象就+1，如果失效了就-1，当为 0 就会回收但是 JVM 没有用这种方式，因为无法判定相互循环引用（A 引用 B,B 引用 A） 的情况。

（2）可达性分析算法（引用链法）： 通过一种 GC ROOT 的对象（方法区中静态属性、常量引用的对象，虚拟机栈中引用的对象等）来判断，如果有一条链能够到达 GC ROOT 就说明不可以回收，不能到达 GC ROOT 就说明可以回收。

6. 强引用、软应用、弱引用、虚引用的区别

强引用：如果一个对象具有强引用，那么垃圾回收期绝对不会回收它，当内存空间不足时，垃圾回收器宁愿抛出 OutOfMemoryError，也不会回收具有强引用的对象。

软引用(SoftReference)：当一个对象只有软引用时，只有当内存不足时，才会回收它；可以和引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收器所回收了，虚拟机会把这个软引用加入到与之对应的引用队列中。

弱引用(WeakReference)：在垃圾回收时，无论内存是否充足，都会回收弱引用对象；可以和引用队列（ReferenceQueue）联合使用。

虚引用：必须和引用队列联合使用；在进行垃圾回收的时候，如果发现一个对象只有虚引用，那么就会将这个对象的引用加入到与之关联的引用队列中，程序可以通过判

断一个引用队列中是否已经加入了虚引用，来了解被引用的对象是否要被进行垃圾回收；虚引用主要用来跟踪对象被垃圾回收器回收的活动。

7. 垃圾收集算法（重点）

分代垃圾回收算法：

首先说一下分代垃圾回收算法，因为收集器基本上都采用分代垃圾收集算法。其实也就是将我们堆空间划分为了一个个不同的区域，新生代，老年代，划分出不同的区域后，垃圾收集器可以只回收某一个区域，新生代垃圾回收、老年代垃圾回收、整个 java 堆和方法区的垃圾回收。这样我们就可以根据各个年代的特点选择合适的垃圾收集算法。

比如在新生代中，每次收集都会有大量对象死去，所以可以选择“标记-复制“算法，只需要付出少量对象的复制成本就可以完成每次垃圾收集。而老年代的对象存活几率是比较高的，而且没有额外的空间对它进行分配担保，所以我们必须选择“标记-清除”或“标记-整理”算法进行垃圾收集。

标记-清除算法：直接释放，将标记的区域中的内存释放，简单高效，但是容易产生大量不连续的碎片

标记-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，每次只使用其中的一块，这一块的内存使用完后，就将还存活的对象复制到另一块去，然后再把使用的空间一次清理掉。这样就使每次的内存回收都是对内存区间的一半进行回收。

不存在内存碎片，浪费内存，空间换时间

标记-整理算法

在清除对象的时候，让存活的对象向一端移动，然后清除掉端边界以外的对象；

8. 类加载过程

类加载的过程包括：**加载、验证、准备、解析、初始化。**其中验证、准备、解析统称为**连接。**

（1）加载：通过一个类的全限定名来获取定义此类的二进制字节流，在内存中生成一个代表这个类的 java.lang.Class 对象。（通过全限定名来加载生成 class 对象到内存中）

（2）验证：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，并且不会危害虚拟机自身的安全。验证 class 文件，包括文件格式校验、元数据验证，字节码校验等。

（3）准备：为类变量分配内存并设置类变量初始值，这里所说的初始值“通常情况”下是数据类型默认的零值。（为这个对象分配内存）

类变量，即静态变量，而不包括实例变量。实例变量会在对象实例化时随着对象一块分配在 Java 堆中。

（4）解析：虚拟机将常量池内的符号引用替换为直接引用。

（5）初始化：到了初始化阶段，才真正开始执行类中定义的 Java 初始化程序代码。初始静态成员并且执行静态代码块。（开始执行构造器的代码，初始静态成员并且执行静态代码块）

9. 常见的类加载器

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。加载的作用就是将 .class 文件加载到内存。

主要有一下四种类加载器:

启动类加载器(Bootstrap ClassLoader)，最顶层的加载类，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）:面向我们用户的加载器，负责加载当前应用类路径 classpath 下的所有 jar 包和类。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

10. 双亲委派模型介绍（了解）

如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载器都是如此，因此所有的加载请求最终都应该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（它的搜索范围中没有找到所需的类）时，子加载器才会尝试自己去加载。