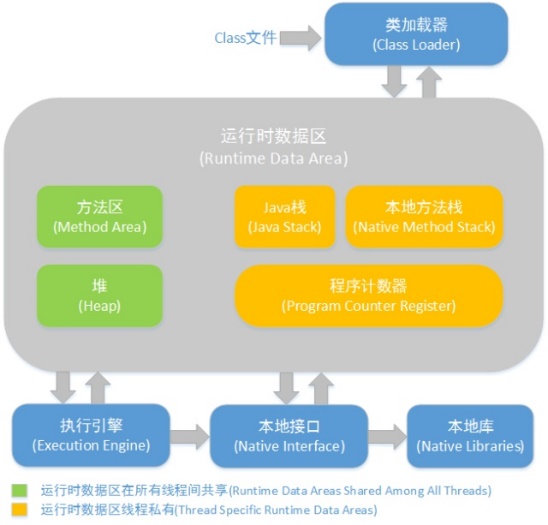
1. 什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？

能够让java字节码文件运行的虚拟机。因为只要是符合jvm规范的字节码文件都能够再上面运行，所以不仅是java能够在jvm上运行，Kotlin等也能在jvm上运行。

1. java内存结构？



程序计数器（线程私有）： 比较小的内存空间，控制着线程的执行顺序，如循环，跳转，异常处理等，还有线程的中止，恢复等。

Java栈/虚拟机栈（线程私有）：java栈对应着方法执行的内存模型，每个方法在执行的时候就会创建一个java栈，他存储着局部变量表，方法出口，操作栈等信息。

本地方法栈（线程私有）：他和java栈作用是相似的，不过java对应的是java的方法，而本地方法栈是对应的本地方法。

Java堆（线程共享）：堆里面存放了java中几乎所有的对象。他也是虚拟机管理的内存中最大的一块。

方法区：存储已经被虚拟机加载的信息，常量，静态变量等数据。

1. 解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法
2. 对象的创建过程
3. 当虚拟机收到New的字节码指令，先会去方法区中寻找，并且检查是否已经被加载过，如果没有被加载，首先执行类的加载过程。
4. 为新生对象分配内存
5. 对对象进行设置，包括对象属于哪个类的实例，GC的年龄信息，元数据信息等。此时所有的数据都是零值
6. 初始化类，执行构造函数，真正的对象被构造出来了。

memory =allocate();    //1：分配对象的内存空间

ctorInstance(memory);  //2：初始化对象

instance =memory;     //3：设置instance指向刚分配的内存地址

1. OOM

OOM是所需要的内存超过申请的内存或者物理内存，也叫做内存溢出。在jvm中，以下几种情况会导致OOM：

1. **Java堆溢出**：当不断地创建对象，并且让GC root和对象之间有可达路径避免被垃圾回收机制清理，那么随着对象地增加，使存储对象的容量超过堆的总容量，则会报错。**OOM:Java head space.**
2. **虚拟机栈和本地方法栈溢出：**1.当栈深度超过虚拟机允许的最大深度，会抛出StackOverFlowError。2.当虚拟机栈内存允许动态扩展，但申请不到足够内存的时候，报OOM错误。（递归或者创建线程）
3. **方法区和运行时常量池（属于方法区）溢出：**方法区中存放着类的类型相关信息，如果加载很多的类，会将类信息放如方法区中，就可能会发生OOM。PermGen space

总结：内存模型中的5大区域，无论哪一个存放的数据所需要的内存超过了限制，都会发生OOM。

1. 如何判断哪些对象可以被回收

两种方式：

1. 引用计数法：在对象中添加一个引用计数器，每当有一个地方引用他，计数器加一，引用失效计数器减一，当计数器为0的时候，就表明不会被使用，会被回收了。但是由于存在循环引用的问题，现在主流的虚拟机都没有采用这种方法，而是采用第二种，可达性分析算法。
2. 可达性分析算法：如果一个对象从GC root，沿着引用链可以被搜索到，那么就是有用的，否则就是不能被使用，可能会被回收：GC root对象：1.方法中的局部变量，临时变量，参数等 2.引用类型静态变量，3.常量引用的对象，如字符串。4.本地方法引用的变量 5.虚拟机内部的引用，如常驻的异常对象（OOMError,NullPointError），基本类型对应的类对象。
3. 引用的类别

4种，分别是强引用，软引用，弱引用和虚引用

**强引用**：普遍存在的引用赋值，如Object a = new Object();永远不会被回收

**软引用**：还有用，但是非必须的对象，在即将OOM之前会被回收。

SoftReference<Object> a = new SoftReference<Object>(); 可以用来实现高速缓存。

**弱引用**：也是还有用，但非必须，比软引用要更弱一点。不会被主动回收，但是发生垃圾回收时就会被回收，无论空间是否够。可以解决HashMap中OOM的问题，将弱引用对象作为键。WeakReference<Object> a = new WeakReference<Object>();

**虚引用**：对对象的生存时间没影响。作用：在被回收的时候收到通知

1. 什么时候进行被回收
2. 当对象实例话的时候查看eden区是否有足够的内存，如果有，直接放在eden区
3. 如果不足，就进行一个minor gc,回收eden区和survior区的对象，然后再判断eden区是否有足够的空间，有则直接放在eden区
4. 如果不足，就把部分存活对象放在survior区，再尝试放在eden区
5. 如果还是不足，向老年代发出请求，如果够，将部分存活对象放入老年区，将新对象放入eden区。
6. 如果还是不足，进行full gc，然后再进行判断
7. 如果还是不足，抛出OOM。
8. 什么时候会被回收

需要被两次标记才会真正被回收。第一次：当被**判定不可达**的时候，会被第一次标记。并加入回收队列中。第二次：当回收时,如果对象没有覆盖**finalize方法**或者已经被调用过，则被视为没有必要执行(覆盖的finalize方法，可以进行逃脱)。

1. 回收方法区

主要回收废弃的常量和不再使用的类型。

废弃的常量：如字符串常量“java”，没有任何字符串值指向该常量，即没有人使用，当垃圾回收时，可能会被回收。

类被回收：1.该类以及子类所有实例被回收。2.加载该类的类加载器被回收,此条件一般很难达成，因为一般是通过基类加载器加载的，只有是替换后的类加载器可能被回收 3.Class对象没有被引用，没有无法通过反射访问该类。满足后，允许被回收，对象是没有引用了肯定会被回收，类需要通过参数进行控制，设置类需要被回收才行。

1. 垃圾收集算法
2. 标记-清除算法

标记（通过GC root无法找到）所有需要回收的对象，统一回收所有被标记的对象.

优点：简单。

缺点：1.执行效率不稳定，当大部分需要被回收时，需要进行大量的标记。2.导致空间碎片太多，从而有大对象无法放入时，提前触发垃圾回收。

1. 标记-复制算法(新生代常用)

内存分为大小相等的两块，每次只使用一半，当回收时，将存活的对象连续复制到另一半。如果空间不足，需要老年代进行分配担保。

不过当时IBM公司曾经研究过，有98%的对象挨不过第一轮回收，因此并不需要1：1的空间，实际中为8:1:1,一个eden，两个survivor，新生代都在eden区，survivor为上一次存活的对象，当进行垃圾回收的时候，将存活的对象放到另外一个空的survivor区中，然后对eden和survivor进行回收，这样只有10%被浪费。不过由于10%，当存活对象超过10%的时候，就需要其他内存区域进行分配担保，通常是老年代，使存活对象直接进入老年代。

优点：无空间碎片

缺点：1.当大量对象存活的时候，需要进行大量内存空间复制的开销。2.浪费了一半的空间。

1. 标记-整理算法（老年代常用）

前面和标记-清除算法一样，不过为了解决空间碎片的问题，在清除之后会对对象进行整理，将对象向内存空间的一端进行移动。

优点：无空间碎片

缺点：移动对象并更新引用极为负担，并需要stop the world。

1. 垃圾收集器
2. 对象分配规则
3. 什么是类的加载
4. 类加载器
5. 类加载原理机制
6. 对象创建过程
7. 类的生命周期
8. java对象结构
9. Java对象的定位方式
10. 引用的分类
11. 回收方法区
12. GC日志分析
13. 调优工具
14. 调优命令

1. JVM性能调优
2. MinorGC和FullGC
3. 内存分配和回收策略