题目 01- 请你用自己的语言向我介绍 Java 运行时数据区

·堆：java堆是java虚拟机给创建的对象分配内存的地方，也是垃圾回收机制的主要区域

·虚拟机栈：虚拟机栈是JVM执行方法的java堆中的内存位置，为线程私有且生命周期与线程的生命周期相同，里面的基本元素称为栈帧，栈帧存储了方法的局部变量、操作数栈、返回地址、动态连接等信息，方法的执行是栈帧的出栈，调用新的方法是栈帧的创建并成为当前栈帧。栈内存大小决定了调用方法的深度。

除了虚拟机栈还有本地方法栈，提供内存空间给JVM的C++方法进行使用

·方法区（永久代、元空间）：方法区是了所有线程共享的内存位置，在java8之前，称为永久代，内存空间在堆内存，在java8之后称为元空间，内存位置在物理内存中。在java1.6，方法区存储了静态变量、字符串运行时常量池、JIT编译数据、类信息。在java1.7中，字符串常量池和静态变量移动到了堆内存中。

·运行时常量池（字符串常量池）：一个class文件拥有1个运行时常量池和class常量池。；一个JVM只有一个字符串常量池，使用了StringTable存储字符串值提高了查询的速度。

·直接内存：直接内存就是位置在java堆外内存，可以提高I/O速度，java堆的内存分配资源比较快，但是I/O效率不高。

题目 02- 描述一个 Java 对象的生命周期

* 1解释一个对象的创建过程

1. 获取到new指令创建对象；
2. 双亲委派机制先自底向上进行类加载验证，看类是否已经加载过了；
3. 双亲委派机制自顶向下进行类的加载器寻找，找到合适的加载器进行加载；
4. 但是对于特殊的场景，比如数据库连接的Drive，会打破双亲委派机制，进行加载。

* 解释一个对象的内存分配

1. 对象分配内存的方式由指针碰撞和空闲列表。指针碰撞代表内存位置是连续的，发生在使用垃圾收集器为Paral和serial，空闲列表代表内存位置不是连续的，被分成了reigon，主要见于CMS收集器；
2. JVM通过CAS和TLAB保证了内存分配的线程安全，TLAB是指JVM会为每一个线程创建的时候通过CAS预先分配内存，如果属于如果TLAB内存耗尽，再通过CAS进行内存的分配；
3. 大多数的对象一开始都会在新生代进行内存的分配，

但是如果对象太大，而且收集器是Serial和Paral收集器，对象就会直接进入老年代。

·解释一个对象的销毁过程

1. JVM通过可达性分析（对象是否被引用）和根搜索算法（从GC root开始向下搜索，找到该对象是否在引用链当中），找到堆中的垃圾对象；
2. 通过2次标记，第一次是通过可达性分析和根搜索算法标记出对象是否不再引用，第二次查看这个对象是否需要被回收。
3. 新生代和老年代都有垃圾回收的机制，主要有标记清除算法、复制算法、标记-整理算法

·对象的 2 种访问方式是什么？

1. 句柄：通过在java栈当中的reference找到句柄池，然后句柄池存储了java对象的指针，然后到实例池子找到对象的实例数据，到方法区找到对象的类型数据
2. 通过直接指针到实例池子找到对象的实例数据，找到方法区对应的对象类型。

* 为什么需要内存担保？

1、java虚拟机是使用分代回收策略，如果经过了minor GC以后新生代的内存依然比较满，就需要老年代来提供内存担保，把survivor区无法分配对象分配到老年代，不然的话新生代很有可能分配内存的时候就占满了，出现OOM的情况会十分频繁。

题目 03- 垃圾收集算法有哪些？垃圾收集器有哪些？他们的特点是什么？

1. 标记-清理算法
2. 标记-整理算法
3. 复制算法

* ParNew 收集器：

年轻代采用复制算法，并行多线程收集；老年代采用标记-整理算法；串行。可指定新生代的收集线程数量，在gc的时候需要进行STW，单cpu性能没那么好因为存在线程的切换。

* ParallelScavenge 收集器：

年轻代采用复制算法，并行收集；老年代采用标记-整理算法；串行，需要STW

* ParallelOld 收集器：

ParallelScavenge 的老年代，使用多线程的标记-整理算法，需要STW

* CMS 收集器：

使用标记-清理算法。可以在java堆达到内存的某一个阈值的时候就开始进行收集

过程分为：第一次标记（STW）、并发标记（不会STW）、重新标记（STW）、并行垃圾清理

* G1 收集器：

面向大内存的应用，官方文档建议少于2G的都不太适合使用该收集器。

把java堆分成多个region，有老年代、新生代和大对象区（用来存储短期使用的大对象）。

总体使用复制算法，老年代回收的时候不会把所有的的region都进行回收，只会对一部分的老年代region进行回收，老年代跟新生代是同时回收的。

过程分为：初始标记（STW）、并发标记（不会STW）、最终标记（STW）、对各个region的回收价值、成本与用户所指定的GC停顿时间进行清理。

但是region存在别的region中的对象引用，尤其是如果老年代被新生代引用，就需要扫描整个java堆才准确，G1通过remember set进行避免了这个问题。每一个region都有一个remember set，每次进行写操作都会产生write barrier。然后检查将要写入的的引用指向是否存在于不一样的region，如果不同，就会通过CardTable把相关引用信息记录到引用指向对象所在的region的remember set