##### 作业题：JVM 虚拟机论述题

作业一定要用自己的语言描述。复制粘贴可不是自己的！感觉自己这么说，能说明白了就达到标准  
了！  
题目 01- 请你用自己的语言向我介绍 Java 运行时数据区（内存区域）

* 堆、虚拟机栈、本地方法栈、方法区（永久代、元空间）、运行时常量池（字符串常量池）、直接内存

答：

堆：jvm运行时用来存储对象的区域。从jdk7开始也用来存储运行时常量池、字符串常量池。是线程共享的。

虚拟机栈：jvm运行时用来存储java方法栈帧的区域。每一个java方法的执行，都意味着一个方法栈帧的入栈到出栈的过程。是线程独有的。

本地方法栈：jvm执行本地方法时用来储存本地方法栈帧的区域。线程独有的。

方法区：jvm运行时用来存储类的元数据（class,filed,method），类的静态变量，类对象的类型指针，常量、字面量，JIT实时编译的代码的区域。线程共享的。在jdk8以前其实现方式是永久代，jdk8及以后采用元空间的方式实现，并把运行时常量池移到了堆内存中。

运行时常量池：一个class对象一份，用来存储类的类型信息。字符串常量池是全局只有一份，用来存储字面量。

直接内存：在jdk1.4，引入了一种非阻塞的新IO流-NIO，可以操作直接内存来提高io操作的效率。直接内存是一种堆外内存，通过直接内存读写效率高于使用堆内内存，但是直接内存的分配性能比使用堆内内存低。

* 为什么堆内存要分年轻代和老年代？

答：java的堆内存设计是基于分代收集理论来实现的。分代收集理论中提出了两种假说：

弱分代假说：大多数对象都是朝生夕灭的，只有少部分对象存活时间较长。

强分代假说：熬过了垃圾回收的次数越多，对象越难以回收。

题目 02- 描述一个 Java 对象的生命周期

* 解释一个对象的创建过程

答：new()指令-->运行时常量池中查找类是否被加载，否则触发类的加载-->堆内存分配空间-->设置零值-->设置必要的信息-->执行init()初始化

* 解释一个对象的内存分配

答：对象的内存分配一般有两种方式。指针碰撞与空闲列表。

指针碰撞：将内存分为已使用和可用两块区域，每一次对象分配内存会移动已使用区域的边界指针，直到指正与未使用边界指针碰撞(可用内存不足或者可用内存使用完)

空闲列表：jvm维护一个列表来记录哪些内存已使用，哪些内存未使用，在对象内存分配时分配到未使用的区域。

内存分配会存在多线程竞争导致内存分配安全问题。Jvm中采用TLAB与CAS的方式来解决该问题：

每个线程会预先分配到一个内存分配缓冲区，线程中的内存分配就不存在安全问题了。当缓冲区使用完毕后，再申请新的缓存区时，采用cas乐观锁的方式来分配内存。

* 解释一个对象的销毁过程

答：当一个对象成为垃圾对象时，会被jvm回收。回收过程中会有两次标记过程。

第一次标记：当jvm判断一个对象为垃圾对象时，会标记该对象为待回收对象。此后会执行且仅执行一次对象的finalize()方法，对象拥有拯救自己的机会。

第二次标记：执行对象的finalize()方法后，对象仍是垃圾对象。Jvm会将其第二次标记，后面会被垃圾回收掉。

* 对象的 2 种访问方式是什么？

答：句柄访问和直接访问。

句柄访问：栈中的局部变量先访问堆中的句柄池，再根据句柄中记录的地址访问堆中的对象。

直接访问：栈中的局部变量直接访问堆中的对象。

* 为什么需要内存担保？

答：内存担保：新对象内存分配时，年轻代中的内存不足，老年代提供担保，将年轻代中年长的对象移动到老年代中，将新对象分配到年轻代中的过程。内存担保可以提高内存的使用率，避免堆内存整体使用率不高，到时却报错oom的问题。

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？

* Serial 收集器

年轻代的一种收集器。采用复制算法，单线程串行的。可以搭配SerialOld、CMS使用。

* ParNew 收集器

年轻代的一种收集器。采用复制算法，多线程并行的。可以搭配SerialOld、CMS使用。

* ParallelScavenge 收集器

年轻代的一种收集器。采用复制算法，多线程的。可以搭配ParallelOld、SerialOld使用

* SerialOld 收集器

老年代的一种收集器。采用标记整理算法，单线程的。

* ParallelOld 收集器

老年代。采用标记整理算法，多线程。

* CMS 收集器

老年代。采用标记清除算法，多线程的，高吞吐量的。优化了STW的时间

* G1 收集器

全堆收集器。采用标记整理算法，局部采用复制算法。基于Region堆内存区域的，低延时的，多线程的，用户可指定STP时间的高效收集器。

* ZGC 收集器

全堆收集器。采用标记整理算法，低时延、多线程。