JVM-Java内存区域与内存溢出异常

一，运行时数据区

- 六个区域
  - 程序计数器
  - 虚拟机栈
  - 本地方法栈
  - 堆
  - 方法区
- 程序计数器(线程私有)
  - 它是一块较小的内存空间，它可以看作当前线程所执行的字节码的行号指示器，由于Java虚拟机的多线程是轮流切换并分配处理器执行的时间的方式来实现的，在任何一个时刻，一个处理器都会执行一条线程中的指令。因此，为了线程切换后能恢复到正确的执行位置，每条线程需要有一个独立的程序计数器，各个线程之间互不影响，独立存储。
- Java虚拟机栈(线程私有)
  - 它的生命周期与线程相同。虚拟机栈描述的是Java方法执行的内存模型：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，方法出口等信息，每个方法从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从入栈到出栈的过程。
  - 两种异常：StackOverflowError异常,OutOfMemoryError异常
    - StackOverfloeError异常：当线程请求的栈深度大于虚拟机所允许的深度是，将抛出该错误。
    - OutOfMemoryError异常：如果虚拟机栈可以动态扩展，如果扩展时无法申请到足够的内存，就会抛出该错误。
- 本地方法栈
  - 与虚拟机栈发挥的作用差不多，只不过虚拟机栈为虚拟机执行Java方法(字节码服务),本地方法栈则为虚拟机用到的Native方法服务。
  - 与虚拟机栈一样，本地方法站包含了两种错误类型：StackOverflowError异常,OutOfMemoryError异常
- Java堆(线程共享区域)
  - Java堆是所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存，但是随着JIT编译器的发展与逃逸技术的逐渐成熟，栈上分配，标量替换优化技术将会导致一些微妙的变化所有的对象都分配在堆上也渐渐变得不是那么绝对了。
  - Java堆是垃圾收集器管理的主要区域，由于收集器采用分代收集算法，所以java堆还可以分为新生代和老年代，再细致一点就是分为Eden空间，From Survivor空间，To Survior空间。从多线程的角度来看，Java堆可能划分出多个线程私有的分配缓冲区。
- 方法区(线程共享区域)
  - 它用故意存储已被虚拟机加载的类信息，常量，静态变量，即时编译器编译后的代码等数据。
  - 垃圾收集行为在这个区域很少出现。
  - 当无法申请到内存的时候，会抛出OutOfemoryError异常。
- 运行时常量池(方法区的一部分)
  - 类编译时期生成的各种字面量和符号引用在类加载之后进入方法区运行时常量池。
  - 抛出的异常情况一方法区一样。

二，实例对象

1，对象的创建过程

- 虚拟机遇见一条new指令之后，首先会检查这个指令的参数是否能在常量池中定位到一个类的符号引用，并检查这个符号引用代表的类是否已经被加载，解析和初始化过。如果没有，则执行类加载的操作。
- 在类加载检查通过之后，接下来虚拟机将为新生对象分配内存。对象所需的内存在类加载的时候就已经确定了。
- 在内存分配完之后，虚拟机将分配到的内存空间都初始化为零值。
- 接下来，虚拟机就要对对象进行必要的设置，例如这个对象是哪个类的实例，对象GC分代年龄等信息。这些信息存放在对象的对象头之中。根据虚拟机当前运行状态的不同，是否启用偏向锁，对象头会有不同的设置方式。
- 在上面的步骤执行完之后，从虚拟机的角度上来看，已经完成了一个新的对象实例的建立，但是从Java程序的视角上来看的话，对象创建才刚刚开始，< init >方法还未执行，所有字段还是零值。一般来说，执行完new命令会执行< init >方法，把对象按照程序员的意愿进行初始化，调用相应的构造函数去初始化，这样一个真正的对象才算完全产生出来。

2，对象的内存布局

对象在内存中主要分为三块区域

- 对象头
  - 用于存储对象自身的运行时数据，如哈希表，GC分代年龄，锁状态，偏向线程ID。
  - 类型指针，即对象指向它的类元数据的指针，虚拟机通过这个指针来确定这个对象是哪个类的实例。
- 实例数据

  这部分是对象真正存储的有效的信息，页式在程序代码中所定义的各种类型的字段内容。
- 对奇填充

  这部分不是必然存在的，也没有特别的含义，仅仅起着占位符的作用。

三，对象的访问定位

分为两种：使用句柄和直接指针。

- 句柄：Java堆中将会划分出一块内存来作为句柄池，句柄中包含了对象实例数据和类数据的地址信息。
- 直接指针：Java堆中对象实例数据中包含了类数据的地址信息。
- 两者的比较
  - 使用句柄最大的好处就是reference(引用)中存储的是稳定的句柄地址，在对象被移动时(GC回收的时候)，只会改变句柄池中实例数据的地址信息，而reference中本身不需要修改。
  - 使用直接指针好处就是速度更快，它节省了一次指针定位的开销(Sun HotSpot就是采用这种方式)。

四，各个区域溢出的情况

- Java堆溢出
  - java.lang.OutOfMemoryError: Java heap space
  - 申请的对象太多了
- 虚拟机栈和本地方法栈溢出
  - 虚拟机栈溢出（大多数情况下达到1000-2000是没有问题）
    - java.lang.StackOverflowError

        - 如果是建立多线程导致的内存溢出，在不能减少线程或者更换64位虚拟机的情况下，就只能减少最大堆和减少栈容量来换取更多的线程。
- 方法区和运行时常量池溢出
    - java.lang.OutOfMemoryError: PerGen space（运行时常量池溢出）