JVM java内存分配和执行

1. JVM是运行在操作系统上的用来执行java的虚拟计算机。包括了指令集，寄存器等等他的目的是用来解释java源文件编译后产生的字节码文件。程序运行时实例化虚拟机，就会开始执行，从而实现跨平台。
2. 线程

线程是程序执行过程中的一个载体，jvm允许执行多个线程。JVM里的java线程和原生操作系统里的线程有直接的映射关系。

在运行java程序时，先创建jvm线程，为它把线程本地存储，缓冲区，同步对象等准备好后创建操作系统原生线程。Java线程结束后，再回收原生线程。

1. JVM内存区域

JVM内存区域可以分为线程私有区域，线程共享区域和直接内存。

线程私有区域包括了程序计数器，虚拟机栈，本地方法栈。他的生命周期和该线程相同，与操作系统本地线程的生与死相对应。

线程共享区域主要包括了方法区和类实例区，这里是所有jvm中的线程都可以访问的，随着虚拟机的启动和关闭而创建销毁。

直接内存：不是运行时数据区的一部分，但会被频繁的使用。避免了java堆和Native堆之间来回复制数据。

1. JVM内存区域的线程私有区域

程序计数器：当前线程所执行的字节码的行号指示器。即当前字节码指令的地址

虚拟机栈：描述java方法执行的内存模型，方法在执行时会创建一个栈帧，存储局部变量表，操作数栈，动态链接，方法出口等。每一个方法从调用到执行完成的过程，就对应了栈帧再虚拟机栈从入栈到出栈。

栈帧随方法调用而创建，随方法结束而销毁。

本地方法区：？不太懂

1. JVM内存区域的线程共享区域

堆，运行时数据区。是被线程共享的，创建的对象和数组都保留在堆中，也是垃圾收集器进行垃圾收集的最重要的区域。

方法区：用于存储被JVM加载的类信息，常量，静态变量，即使编译器编译后的代码等数据。运行时常量池也是方法区的一部分。

1. JVM运行时内存，JVM堆

JVM堆是存放各种对象数组的区域，是全剧共向的内存区域，也是垃圾收集器最关心的进行垃圾回收的区域。因为目前采用分代收集算法，所以jvm堆也可以细分成新生代，老年代。

新生代：

这里的对象是朝生夕死，不太稳定的。分为eden区，s1，s2，一般eden区占80%，s各占10%。在这里进行的垃圾回收算法是MinorGC，它采用复制算法。每一次执行都是把eden区和S1区的内容，对象拷贝到S2区，并且年龄加一，然后删除原来内容。如果有对象年龄达到了老年标准，就赋值到老年代区。最后将s1，s2互换。

老年代：这里执行的垃圾回收算法是MajorGC，因为他比较稳定。一般在新生代晋升老年代或者要分配较大的新创建对象时才会触发MajorGC来垃圾回收，腾出空间。

MajorGC采用的是标记清除算法，扫描一次老年代，标记出存活的对象，回收没有标记的对象，耗时久。这也是我们用新生代的原因。

1. 垃圾回收和算法
2. 如何确定垃圾
3. 引用记数法

当一个对象被创建引用时，设为一，当他超过了生命周期，或者被删除消亡时，减一。当一个对象再没有引用时，可回收。

缺点在于无法解决循环引用。

1. 可达性分析

以每一个GCroot作为根节点，去向下遍历树，寻找可达的点。不可到达的就认为是需要回收的。

1. 回收垃圾算法
2. 标记清除算法

分为标注阶段和清除阶段：标注节点标记出所有需要回收的对象，清楚阶段回收被标记对象占用的空间。

会造成内存碎片化。

1. 标记移动算法

牺牲了一部分效率来换取空间。标注回收之后，所有的内容向左移动，解决内存碎片问题。

1. 复制算法

每次只用启动一块，一块满了后移到另一块上。

1. 分代收集算法

正如上面分代时介绍过的一样，新生代用copy，老年代用标记移动。

1. java四种引用类型

强引用

一般我们写的都是强引用，只要他还存在，就不会回收他。

软引用

弱引用

虚引用

内存泄漏：强引用，线程池，监听器，单例模式，没及时释放引用。

1. GC垃圾收集器

新生代的有：Serial，ParNew，Parallel Scavenge等等

Serial：单线程复制算法垃圾收集器

ParNew：就是Serial的多线程版本

Parallel Scaveng收集器：也是多线程复制算法，不同在于他关注程序达到可控制的吞吐量，同时采用了自适应调节策略。

老年代的有：CMS，MSC，Parallel Old等等

Serial Old收集器：单线程，即CMS，采用了标记整理算法。

Parallel Old：多线程。

CMS：四步：初始标记，并发标记，重新标记，并发清除。

G1收集器：最前沿的东西。

Copy用**Scavenge GC**

对整个堆进行整理 Full GC。可能因为以下原因：

1. 年老代满了
2. 持久代满了
3. Gc清除被显式调用
4. 上一次Gc后堆上的各域分配策略动态变化
5. Java IO/NIO

IO模型分类：

阻塞IO模型：等数据就绪才能接着执行。

非阻塞IO模型：不断询问数据是否就绪，消耗CPU

多路复用IO：通过一个线程不断地去轮询socket状态，当socket真正有读写事件时，再真正调用实际的IO读写操作。

信号驱动IO模型。

异步IO模型：和上一个区别在于信号驱动模型表示数据准备好了，而异步IO模型表示IO操作已经完成了。

JavaIO包分为字符流，字节流。字符流针对普通文件，字节流针对二进制文件。

而JAVANIO则有三大核心部分：通道，缓冲区，选择区。和传统IO不同，传统IO总是针对流的，而NIO是针对缓冲区和通道的。总是从通道到缓冲区或者从缓冲区到通道。是非阻塞的。

Java也提供了一系列包，如channels，charset，buffer，byteorder，mappedByteBuffer等等。

Channel。

Buffer。

Selector。

1. jvm类加载机制

Jvm类加载机制分为五个部分：

加载：在内存中生成代表这个类的对象，作为各种数据的入口。

验证：确保字节流中的信息符合虚拟机当前的要求，并且不会危害虚拟机自身安全。

准备：为变量分配内存空间，在方法区中。

解析：将常量池中的符号引用替换为直接引用。

初始化：执行类构造器方法的过程。

不执行类初始化的场景：

什么时候对类初始化：

类被主动引用或者被动引用

1. 通过子类引用父类的静态字段，只会触发父类的初始化，而不会触发子类的初始化。

2. 定义对象数组，不会触发该类的初始化。

3. 常量在编译期间会存入调用类的常量池中，本质上并没有直接引用定义常量的类，不会触

发定义常量所在的类。

4. 通过类名获取 Class 对象，不会触发类的初始化。

5. 通过 Class.forName 加载指定类时，如果指定参数 initialize 为 false 时，也不会触发类初

始化，其实这个参数是告诉虚拟机，是否要对类进行初始化。

1. 通过 ClassLoader 默认的 loadClass 方法，也不会触发初始化动作
2. 类加载器

启动类加载器。扩展类加载器。系统应用类加载器。

每一个在被使用的时候都会先看看父亲能不能完成，不能完成才用它。这样保证了object的统一。