图文 070、第10周答疑汇总

详情 评论

第10周答疑汇总

学员总结：

总结一下老师这几天提交的参数：

-XX:+CMSParallelInitialMarkEnabled表示在初始标记的多线程执行，减少STW；

-XX:+CMSScavengeBeforeRemark：在重新标记之前执行minorGC减少重新标记时间；

-XX:+CMSParallelRemarkEnabled:在重新标记的时候多线程执行，降低STW；

-XX：CMSInitiatingOccupancyFraction=92和-XX:+UseCMSInitiatingOccupancyOnly配套使用，如果不设置后者，jvm第一次会采用92%但是后续jvm会根据运行时采集的数据来进行GC周期，如果设置后者则jvm每次都会在92%的时候进行gc；

-XX:+PrintHeapAtGC:在每次GC前都要GC堆的概况输出

回答：总结的非常好

==========

问题：

这问题看似简单，其实我也碰到过，当时使用jxl导出excel的时候，jxl会默认调用gc方法，当时花了不少时间才发现原来是这个问题...

回答：

是的，就是如此，还有的同学一时手痒会自己去执行这行代码，就怕有的人自己瞎玩儿

==========

问题：

果然精短轻松?，之前我还在想这个参数的意义在哪里。老师，像这种情况jstick里会有什么异常吗?

回答：

jstat其实是用在死锁里的，一般可以是看死锁哪里线程卡住的

==========

问题：

老师，永久带里面存放的类信息具体是一些什么样的信息，可以给讲解几种吗？谢谢

回答：

这个具体类的信息，其实你可以理解为就是类所带的一些变量和方法

==========

总结：

当多个服务需要竞争一个单体资源时，可以考虑加上分布式锁。如果并发量高的话，可以考虑拆分掉那个单体资源，50个拆成5个10个资源，从而缩小锁的粒度，提高吞吐量。

总结的很好

==========

总结：

一般redis的分布式锁都可以使用redisson框架来做。 使用：直接lock("key")

原理：当lock的时候，当前客户端会生成一串lua脚本发送到redis服务端。服务端根据这个key是否存在以及key的value状态判断是否已被加锁。

如果加锁成功（有效期30s）：生成的value里面带有当前客户端的id，实现可重入效果。然后在这个线程执行过程中，一旦加锁成功还会有一个watch dog每10s去刷新锁的状态。直到释放或者宕机，如果当前线程死锁或阻塞，那么这个锁就一起死锁。如果加锁失败：自旋到获取锁。

总结的很好

==========

问题：

老师，我有个关于锁的问题。在我看来，所有锁都是基于一个互斥量，这个互斥量可以随意选取为一个支持加锁或者原子操作的第三方组件。那么经常使用zk/redis去是实现分布式锁，而不使用db去加锁。是因为db的吞吐量太低了吗，还是有其他考虑？

回答：

db不适合配合系统做加锁，从功能层面，db就不是干这个的

==========

学员思考题回答：

出现原因：

因为redis主从异步复制的实现方式是可能出现在主节点宕机时未完成同步而出现的锁丢失。 这个时候watch dog发现了也没办法，只能说是自己停掉了。

解决办法：

因为主从异步复制出现的问题，那么保证主从写强一致就可以了。但是如果写强一致的话，那么吞吐量就会因为这些实时同步而急剧降低。

另外出现了C，那么A这个可用性就会受到一定的影响，因为在实时同步的时候，要是失败了，这个服务可能暂时就无法提供正常的服务。

回答：

总结的很好

==========

总结：

zk一般使用curator去实现分布式锁。 原理：向zk发起请求，在一个目录（/locks/pd_1_stock）下，创建一个临时节点，也是带有自己客户端的id。

如果目录是空的，自己创建出来的节点就是第一个节点，那么加锁成功。如果成功执行则释放（节点删除）。

如果宕机了，基于zk的心跳机制，那个临时节点也会被删除。第二个客户端请求锁时，也创建一个节点，如果不是第一节点，那么向上一节点加一个watcher监听器。如果上一节点被删除立马会感知到，然后在判断自己是不是第一节点，如果不是再监听上一级（公平实现）。完事后陷入等待，直到获取锁。

总结的很好

==========

总结：

羊群效应：当几十个节点争抢同一把基于zk的锁时，如果都是监听第一个节点，那么当释放锁时，zk会同时反向通知所有客户端又来重新增强。

影响：

主要是多了很多没必要的请求，从而会加重网络的负载。

解决：

就基于curator去做就好了，通过监听上一级节点，降低了争抢次数，还实现了公平锁。

redis：redis因为是客户端自己主动隔一段时间去尝试加锁，所以羊群效应影响不大，因为请求都错开了，而不是一群一拥而上。

总结的很好

==========

问题：

老师，关于g1我有个疑惑，就是如果选择减小gc耗时，那是不是意味gc的次数会增多。那选择减少时间是因为它的收益大于gc次数的增加。

回答：

是的，你理解的没问题，gc频率多点，但是每次gc耗时很短，没事的

==========

问题：

老师能不能对 这种类型的问题上一课 最近金九银十 肯定有很多同学要面试 比如这个题目 凌晨三点发生了一次oom 怎么追踪解决

回答：

这个问题很好，这个在后面的OOM相关案例里会分析的

==========

问题：

老师好，刚才看到有同学的总结：
G1，新生代未达到60%，老年代未达到45%，则不进行GC 您的回答是没问题，但是从您的文章中找到的，新生代不到60%，只要ygc时间接近预设GC时间时就会进行GC。
看了一下我们的线上系统，发现ygc之后，survivor区已经达到了将近90%，但是没有对象进入老年代，这是不是意味着要么所有对象都是同岁的，要么至少41%的对象是大年龄且同岁的。

回答：

有的回复可能是没看清楚他们的问题，G1的话，有可能达到预期的gc时间后他就会触发gc，不是固定的

==========

问题：

看到老年代驻留大量对象，想到的竟然是老年代空间太小，需要扩大，而不是其他问题，看来还是没学到家呀

回答：

是的，所以要多学案例，积累经验

==========

问题：

打卡，今天讲解了cpu负载高的原因，复习了频繁fullGC的原因，给了一个分析堆内存的实用工具MAT。给力。

回答：

是的，今天其实以复习为主

==========

问题：

老师我非常激动，今天出现一个问题，我发现和上周的案例基本一致，就是极端情况下，SQL中in条件语句包含十几万个ID，一个SQL语句的String对象很大，导致Full GC，然后还有一种场景in条件是空 ，SQL是select * from DB的(代码我写的，非常惭愧)一下加载十几万的数据出来，导致full gc。

除了这两个问题，今天的例子也和我们系统很想：通过cat观察cms管理的内存平时一般都在800m左右，但是最近一直涨到1300m左右，持续了很久都不会降低，老年代2000m，68%就会触发Full GC，也就是1300左右就会发生GC，我初步怀疑是哪里内存泄漏了，导致这里占了几百MB内存不释放，这个还有待查询。准备将这个泄漏找出来，然后调整一下JVM参数，优化一波。

回答：

非常好，看来案例中碰到的例子，你自己工作里都碰到了，一个是SQL误操作加载过多数据触发的Full GC，一个是内存泄漏导致老年代里一直占用了过多内存。可以dump一个内存快照出来，用MAT分析一下

==========

问题：

另外有点疑问，大对象进入老年代，我网上找资料说是默认大对象阈值是0，也就是不会直接进入老年代，这个是真的吗。

我感觉我的问题也是因为很大的String直接进入了老年代才导致几百MB无法释放

回答：

大对象会直接进入老年代的，但是具体要看不同版本的JDK默认参数是什么

==========

问题：

老师，你好，我数据平台有一个报表导出，导出的时候因为堆内存不足而失败，但是后面CMS，会一直进行标记清理，老年代基本上是占满的，S1,S2没有任何占用，jstat 看到jvm一直在进行 YGC 和 FGC

但是老年代一直不减少，只有在eden区占满以后执行后的YGC，才会将老年代的占用真正的回收，请问能否解释一下这个是为什么?

-XX:InitialHeapSize=1073741824 -XX:MaxHeapSize=1073741824 -XX:NewSize=838860800 -XX:MaxNewSize=838860800 -XX:MetaspaceSize=104857600 -XX:SoftRefLRUPolicyMSPerMB=2000 -XX:SurvivorRatio=2 -XX:MaxTenuringThreshold=15 -XX:PretenureSizeThreshold=20971520 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=95 -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 -XX:+CMSParallelInitialMarkEnabled -XX:+CMSScavengeBeforeRemark -XX:+TraceClassLoading -XX:+TraceClassUnloading -XX:+PrintGCDetails -XX:+PrintGCDateStamps -verbose:gc -Xloggc:./filebeat/gc.log -XX:ErrorFile=./logs/jvm_error.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=./logs/heapdump.hprof 204800.0 204800.0 0.0 0.0 409600.0 313375.4 229376.0 229352.0 ﷺ 252 1.854 467 54.764 56.618 还有就是元数据区是乱码的

回答：

这个本质还是你代码的问题，应该是大量的数据占满了老年代，没处理完就回收不掉

==========

问题：

请问一下[Loaded sun.reflect.GeneratedMethodAccessor129 from __JVM_DefineClass__]从这行日志是咋看出来GeneratedMethodAccessor会被加载到MetaSpace的, 这行日志好像没有提供类被加载到哪里的信息

回答：

第一个英文单词，Loaded，就是加载的意思

==========

总结：

cms存在的问题，浮动垃圾因为并发清除，空间碎片因为标记整理算法，并发执行失败也是因为并发清除的设计可能存在预留的老年代空间不足。但是cms对空间碎片进行了优化，提供了内存的整理，这个操作可以通过参数去控制，默认是开启，并且每次fullgc后都去整理内存，但是就需要stw。

最后，fullgc的发生情况：

1.老年代可用内存小于新生代全部对象的大小，又没有开启空间担保，就会直接触发fullgc。

2.如果新生代存活大小大于老年代空间，并且老年代空间小于历次晋升的平均内存大小，也会执行fullgc。

3.大对象或者动态年龄进入老年代，而老年代空间不足，也会执行fullgc。

4.如果是cms回收器，那么老年代内存使用到92%之后，就会触发fullgc，因为并发清除阶段需要给用户线程预留内存空间。

回答：

总结的很好

==========

问题：

一般公司线上系统是禁用dump内存快线的吗。我们被运维禁了

回答：

是的，线上机器一般来说会禁止执行dump，因为dump的时候可能会导致系统停机几秒钟，或者几百毫秒，可以跟运维沟通，让他们给你dump一个快照出来

==========

问题：

打卡。MAT好用，但是功能多需要慢慢摸索

回答：

是的，非常好用，不过文章里介绍的几个常用功能基本够用了，其实一般主要就是看占用内存最多的对象，追踪一下哪个线程，然后看一下线程执行堆栈

==========

问题：

老师您好！我最近遇到一个问题：

有一个定时任务，每5秒去读取一张表中的数据，这张表大概有7000多行，每行大概3KB

理论上，加载到内存，最大差不多30MB左右，但我通过jstat发现，到底每5秒钟，加载了大概300MB的数据，我对此很困惑，望老师或者其他同学能够解答一下～

回答：

这个很正常的，因为实际加载到内存里，你有不同的数据结构来存储，还有一些对象本身也会占据内存空间，实际内存消耗往往比你数据本身大小要大很多

==========

总结：

JVM是什么 答：本质上也是一个程序，负责运行java编译好的字节码文件(.class)。
JVM跟我们平时运行在机器上的系统之间是什么关系 答：JVM具有跨平台性，可以在多种系统上执行。
类加载器的概念 答：把编译好的那些".class"字节码文件给加载到JVM中。
字节码执行引擎的概念 答：针对加载进内存的类进行代码的执行。

回答：

总结得很好

==========

问题：

现在不是很清楚年轻代的gc root跟踪和老年代的gc root跟踪，是不是都是从gc root去找引用的对象？还是年轻代的直接从gc root遍历，老年代得并发标记，从老年代中的对象去找gc root

回答：

都是从gc roots开始追踪的

==========

问题：

老师你好，我在动态年龄判断那块有点疑问，假如survivor内存100m，现在已经占用了45m,那么此时来了一共55m的对象，那么之前所有的对象全部进入老年代，但是这55m还是大于一半，假如都是1岁会怎么处理呢，按对象的创建时间进入到老年代吗？

回答：

对的，也有可能会进入老年代

==========

问题：

老师好，关于动态对象年龄判断，我理解的还是不清晰，我现在理解的是判断各个年龄段之和是不是超过了survivor的一半，但是“各个年龄”包含本次gc之后存活的对象吗，如果超过了一半，那么本次gc之后的对象会进入老年代吗

回答：

其实关于这里的实现，不用过于找里面的细节，因为不同JDK版本实现都不同。你只要知道几点

第一，有动态年龄判定规则

第二，有可能因为young gc后存活对象过多导致进入survivor对象太多，触发这个规则

第三，可能因为这个规则导致很多对象进入老年代

==========

问题：

学这个专栏，感觉就在上课，不断学习，不断的复习

回答：

是的，一味的讲新内容往前赶进度是不靠谱的，要不断的有新的知识，同时不断带着之前旧的知识反复强化，还要定期复习，最后跟着看完，对核心知识绝对都消化的很彻底，当然自己也得注意复习和理解，还有结合自己的实际工作情况去积累

==========

学员思考题回答：

1.调整s区到能够装下每次minorgc后存活的对象。

2.调整动态年龄判断，可以考虑减小默认值。

3.设置大对象进入老年代的判断，可以考虑设置为1m。

4.设置回收器组合，一般响应优先就使用parnew+cms

回答：总结的很好

==========

问题：

系统给了3g堆内存，但是dump出来的文件打开看只有500mb，这个怎么解释

回答：

这个很正常，说明你的内存占用并没有占用满3g，就使用了500mb左右的内存，另外一个dump快照也会做一些二进制存储，不一定跟3g一样

==========

总结：

关于老年代的优化，都是建立在新生代已经进行优化的前提下。然后分析什么情况下对象依旧会进去老年代，多久会填满老年代，再根据垃圾回收器的特性，去设置对应的参数。

一般来说老年代的参数保持默认就好了，新生代优化好了，很少甚至几乎不会发生fullgc。

回答：

总结的很好

==========

问题：

老师，我发现一次young gc并没有将Eden区清空，而是会有部分保留是怎么回事呢？是我弄的不对还是有什么说道的？

回答：

不是的，实际上你说的那种是他刚ygc完，然后就有对象进入了eden，所以你就看到那样子了

==========

总结：

g1是以垃圾回收优先的回收器，主要是通过将内存区域划分成region然后维护一个回收价值列表，建立一个可预测的时间停顿模型。
内部保留了年代，但是仅仅是逻辑上进行了保留，新生代和老年代会根据需要进行变化。

回答：

总结的很好

==========

总结:

rabbitmq丢失数据的情况：

1、生产者端。消息因为网络问题丢失或者发送到rabbitmq时出错了。

2、rabbitmq服务端。未做持久化。

3、消费者端。打开了autoAck，在未完成消费之前就自动回复了。

rabbitmq丢失数据的解决：

1、生产者端。

通过confirm模式异步确认消息发送成功，在失败后的回调函数中处理失败的逻辑。

2、服务端。

打开持久化机制。这里涉及到两个参数，一个是建立queue的时候，持久化那个queue。

另外一个是生产者发送消息的时候，把deliveryMode设置为2，让MQ把这条数据也给持久化。

但是尽管如此，如果在极端情况下，在rabbitmq中内存写成功，但是还没来及持久化时，rabbitmq宕机，这部分在内存里面的数据也会丢失，不过几率很小。

3、在消费者端，去掉autoAck，在自己完成逻辑后手动提交ack。

回答：总结的很好

==========

总结：

kafka出现数据丢失的情况：

1、生产者端：和rabbitmq类似，如果没能确认写成功，也没有重发那么也会丢失。

2、服务端：如果未来得及和从节点同步数据就宕机了，那么这部分数据就会丢失。

3、消费者端：和rabbitmq类似，如果自动提交offset依旧会出现丢失。

kafka出现数据丢失的解决：

1、生产者端：设置参数，要求每个从节点都写成功后才任务成功，另外如果发送失败，重试次数设置一个很大的值。

2、服务端：设置参数，要求从节点起码大于1，且至少有一个能被感知到。

3、消费者端：取消掉自动回复。 不过，强一致的保证消息不丢失，必然会影响到吞吐量。

回答：

总结的很好

==========

问题：

打卡，jvm性能优化怎么做？用jstat呀，哈哈，神器在手，再来个mat工具，还找不到问题？妖怪吧。

回答：

是的，基本gc日志+jstat+MAT就足够了

==========

6.JVM在什么情况下会加载一个类

答:

a.JVM进程启动之后，代码中包含"main()"方法的主类一定会被加载到内存。

b.执行"main()"方法代码的过程中，遇到别的类也会从对应的".class"字节码文件加载 对应的类到内存里面。

7.一个类从加载到使用，一般会经历哪些过程

答：加载->验证->准备->解析->初始化->使用->卸载

a.加载:将编译好的".class"字节码文件加载到JVM中

b.验证:根据JVM规范，校验加载进来的".class"字节码文件

c.准备:给类和类变量分配一定的内存空间，且给类变量设置默认的初始值(0或者nul)

d.解析:把符号引用替换为直接引用的过程

e.初始化:根据类初始化代码给类变量赋值

注：执行new函数来实例化类对象会触发类加载到初始化的全过程；或者是包含"main()"方 法的主类，必须是立马初始化的。如果初始化一个类的时候，发现他的父类还没初始化， 那么必须先初始化他的父类。

8.Java里有哪些类加载器

答: a.启动类加载器: 主要负责加载我们在机器上安装的Java目录(lib目录)下的核心类库

b.扩展类加载器：主要负责加载Java目录下的"lib/ext"目录中得类

C.应用程序类加载器：主要负责加载"ClassPath"环境变量所指定的路径中的类，大致 可以理解为加载我们写好的java代码

d.自定义类加载器：根据自己的需求加载类

9.什么是双亲委派机制

答：JVM的类加载器是有亲子层级结构的，启动类加载器最上层，扩展类加载器第二层，应用程序类加载器第三层，自定义类加载器第四层。

当应用程序类加载器需要加载一个类时， 他会先委派给自己的父类加载器去加载，最终传导到顶层的类加载器去加载，但是如果父类加载器在自己负责加载的范围内，没找到这个类，那么就会下推加载权利给自己的子类加载器

回答的很好

==========

问题：

老师能说说RememberSet吗，我在深入理解JVM这本书上看到了老年代引用新生代对象的时候，是用RememberSet来避免全堆扫描，那进行Old GC的时候，新生代引用老年代，是直接扫描整个新生代吗？我看有的博客有说道CMS的并发预清理跟可中断并发预清理，这又是什么。

回答：

这是一些JVM底层源码级别的原理，我们这个专栏定位是实战型的，基本把日常生产实战的内容都分析过了，而且都是结合案例的。有一些是属于深入底层的技术细节，但是跟平时实战优化关系不大，所以实际上我们没有讲这些内容。也许未来会出那种JVM源码级别的深入的课程，会说到你说的那些内容。

==========

问题：

你的总结正是我心中想总结的，预料之中。jvm运行原理、什么时候yong gc、什么时候进入老年代、什么时候full gc、为什么会频繁full gc、分析制定jvm模板

回答：

是的，定期复习和总结非常的重要

==========

问题：

之前也看过jvm的书，看了一些视频，但怎么优化还是一头雾水。很幸运遇到了老师的专栏，现在不管对实战还是面试都充满信心。

每天都要花3-4个小时学习总结，现在看文章还算轻松，后续要把老师所有的案例总结成自问自答的面试题。也根据所学，基于jstat初步做了个监控小工具，后面继续完善，也算是对专栏最好的总结。谢谢老师！！！！！！

回答：

好的，你真是太棒了，你们学有所成，真正掌握JVM的优化实战，提升国内工程师的技术水平，就是我们希望做的事情

==========

问题：

老师好，线上也出现了FUll GC的告警，日志如下：

2019-09-05T17:26:15.161+0800: 85779.869: [Full GC (Metadata GC Threshold) 2019-09-05T17:26:15.161+0800: 85779.869: [CMS: 472256K->445559K(3022848K), 2.0333919 secs] 1425388K->445559K(4910336K), [Metaspace: 277217K->277217K(1511424K)], 2.0355295 secs] [Times: user=2.01 sys=0.01, real=2.03 secs] 2019-09-05T17:26:17.197+0800: 85781.905: [Full GC (Last ditch collection) 2019-09-05T17:26:17.197+0800: 85781.905: [CMS: 445559K->382990K(3022848K), 1.6770458 secs] 445559K->382990K(4910336K), [Metaspace: 276037K->276037K(1511424K)], 1.6863552 secs] [Times: user=1.63 sys=0.01, real=1.68 secs] 2019-09-05T17:26:18.886+0800: 85783.594: [GC (CMS Initial Mark) [1 CMS-initial-mark: 382990K(3022848K)] 382992K(4910336K), 0.0134842 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]

初步确认是每次发布groovy脚本，classload的时候才会彪高，这种问题基本定位到了，但是需要怎么去优化和解决呢?

回答：

这个很明确了，你的metaspace太小了，所以classload太多的时候会触发full gc，只要给metaspace区域更大空间就可以了

==========

问题：

这个课程真的是我学习过的JVM课程和看过的书中最好的,没有之一,其他的课程都感觉是隔靴搔痒,这个课是直击要害,而且每个问题都能得到回复, 非常感谢

回答:

多谢你的支持

==========

问题:

现在很渴望去解决线上问题，当别人搞不定的时候我来上，可惜现在系统的量太少，基本上堆内存6G,老年代和年轻代分别3G，就不会产生性能问题。只能假设它的量增加100倍了。

回答:

是的，因为系统量太小，就是Eden区都慢慢才满，满了以后存活对象也很少，几乎很少对象进入老年代的，所以一般量小的系统没有JVM的问题

==========

问题:

请问一下如果访问量增加100倍, 比如QPS从100, 猛增到10000, 而我Tomcat的MaxThread设置的是200, 而我每个请求的处理时间加入是100ms, 那理论上系统1S中可以处理的请求数就是2000, 这已经确定了啊, 即使QPS达到10000,多余的请求只会在OS中的Epoll模型中排队吧,

我感觉好像不会轮到JVM的问题暴露, 所以优化jvm参数的时候只要考虑到最大QPS是2000就可以了, 请问我这样理解对不对?

回答:

我给你举个例子怎么思考，比如你现在有一个系统，部署在一台机器上，机器能承载的极限并发量是每秒处理1000个请求，结果你现在系统量很小，每秒才处理10个请求。此时你直接假设你的机器资源被打满，每秒处理1000个请求，同样的内存下，请求量大了1000倍，你的JVM会如何运行?

==========

问题:

今天是分析而不讨论。让我们去回顾之前所学的知识，然后结合自己现有系统去准备这方面的面试技巧。

回答:

是的，加油好好复习和准备，多结合自己工作的系统去思考JVM的优化，放大100倍压力思考一下

==========

11.什么是JVM内存区域划分

答：加载进来的类信息，需要放在某个内存区域。方法运行时，方法里面很多变量之类的 东西需要放在某个内存区域。代码里创建的对象，也需要内存空间来存放。所以JVM中必须 划分出来不同的内存区域，为了让我们写好的代码在运行过程中更方便的根据需要来使用。

12.JVM中有哪些内存区域 答：

a.方法区: 在JDK1.8以后,这块区域的名字改了,叫做"Metaspace"。主要存放我们自己写的 各种类相关的信息。

b.程序计数器：字节码指令通过字节码执行引擎被一条一条执行，才能实现我们写好的代码 执行的效果。

程序技术器就是用来记录当前执行的字节码指令的位置，也就是记录目前执行到了哪一条字节码指令。

JVM是支持多个线程的，所以就会 有多个线程来并发执行不同的代码指令，因此，每个线程都会有自己的一个 程序计数器，专门记录当前这个线程目前执行到了哪一条字节码指令。

c.Java虚拟机栈：保存每个方法内的局部变量等数据。每个线程都会有自己的Java虚拟机栈。

如果线程执行了一个方法，就会对这个方法调用创建对应的一个栈帧(栈帧 里就有这个方法的局部变量表,操作数栈,动态链接,方法出口等)，然后压入 线程的Java虚拟机栈。方法执行完毕之后就从Java虚拟机栈出栈。

因此，每个线程在执行代码时，除了程序计数器以外，还搭配了一个Java虚拟机 内存区域来存放每个方法中得局部变量。

d.Java堆内存：存放我们在代码中创建的各种对象。对象实例里面会包含一些数据。而Java 虚拟机栈的栈帧局部变量表里面的对象，其实是一个引用类型的局部变量，存放 了对应Java堆内存对象的地址。可以理解为局部变量表里的对象指向了Java

回答：

总结得很好

==========

问题：

老师，那以后市面上都用G1清理，那我们是不是在jvm优化上就得靠边站了?

回答：

是的，使用G1的时候，其实能做的事情很少，因为你想，他所有的内存分配和GC时机都是动态变化的，你怎么去调优？实际上他一切都是自动运行的。只要他能保证每次GC的耗时在你指定范围就可以了。但是其实现在G1也未必就已经很稳定了，所以一般还是用CMS+ParNew就可以了，比较可控一些。