14 | 内存管理：如何避免内存溢出和频繁的垃圾回收？

李玥 · 消息队列高手课

你好，我是李玥。今天，我们来聊一聊内存管理的问题。

不知道你有没有发现，在高并发、高吞吐量的极限情况下，简单的事情就会变得没有那么简单了。一个业务逻辑非常简单的微服务，日常情况下都能稳定运行，为什么一到大促就卡死甚至进程挂掉？再比如，一个做数据汇总的应用，按照小时、天这样的粒度进行数据汇总都没问题，到年底需要汇总全年数据的时候，没等数据汇总出来，程序就死掉了。

之所以出现这些情况，大部分的原因是，程序在设计的时候，没有针对高并发高吞吐量的情况做好内存管理。要想解决这类问题，首先你要了解内存管理机制。

现代的编程语言，像 Java、Go 语言等，采用的都是自动内存管理机制。我们在编写代码的时候，不需要显式去申请和释放内存。当我们创建一个新对象的时候，系统会自动分配一块内存用于存放新创建的对象，对象使用完毕后，系统会自动择机收回这块内存，完全不需要开发者干预。

对于开发者来说，这种自动内存管理的机制，显然是非常方便的，不仅极大降低了开发难度，提升了开发效率，更重要的是，它完美地解决了内存泄漏的问题。是不是很厉害？当年，Java 语言能够迅速普及和流行，超越 C 和 C++，自动内存管理机制是非常重要的一个因素。但是它也会带来一些问题，什么问题呢？这就要从它的实现原理中来分析。

自动内存管理机制的实现原理

做内存管理，主要需要考虑申请内存和内存回收这两个部分。

申请内存的逻辑非常简单：

1. 计算要创建对象所需要占用的内存大小；
2. 在内存中找一块儿连续并且是空闲的内存空间，标记为已占用；
3. 把申请的内存地址绑定到对象的引用上，这时候对象就可以使用了。

内存回收的过程就非常复杂了，总体上，内存回收需要做这样两件事儿：先是要找出所有可以回收的对象，将对应的内存标记为空闲，然后，还需要整理内存碎片。

如何找出可以回收的对象呢？现代的 GC 算法大多采用的是“标记 – 清除”算法或是它的变种算法，这种算法分为标记和清除两个阶段：

> 标记阶段：从 GC Root 开始，你可以简单地把 GC Root 理解为程序入口的那个对象，标记所有可达的对象，因为程序中所有在用的对象一定都会被这个 GC Root 对象直接或者间接引用。
>
> 清除阶段：遍历所有对象，找出所有没有标记的对象。这些没有标记的对象都是可以被回收的，清除这些对象，释放对应的内存即可。

这个算法有一个最大问题就是，在执行标记和清除过程中，必须把进程暂停，否则计算的结果就是不准确的。这也就是为什么发生垃圾回收的时候，我们的程序会卡死的原因。后续产生了许多变种的算法，这些算法更加复杂，可以减少一些进程暂停的时间，但都不能完全避免暂停进程。

完成对象回收后，还需要整理内存碎片。什么是内存碎片呢？我举个例子你就明白了。

假设，我们的内存只有 10 个字节，一开始这 10 个字节都是空闲的。我们初始化了 5 个 Short 类型的对象，每个 Short 占 2 个字节，正好占满 10 个字节的内存空间。程序运行一段时间后，其中的 2 个 Short 对象用完并被回收了。这时候，如果我需要创建一个占 4 个字节的 Int 对象，是否可以创建成功呢？

答案是，不一定。我们刚刚回收了 2 个 Short，正好是 4 个字节，但是，创建一个 Int 对象需要连续 4 个字节的内存空间，2 段 2 个字节的内存，并不一定就等于一段连续的 4 字节内存。如果这两段 2 字节的空闲内存不连续，我们就无法创建 Int 对象，这就是内存碎片问题。

所以，**垃圾回收完成后，还需要进行内存碎片整理，将不连续的空闲内存移动到一起，以便空出足够的连续内存空间供后续使用。**和垃圾回收算法一样，内存碎片整理也有很多非常复杂的实现方法，但由于整理过程中需要移动内存中的数据，也都不可避免地需要暂停进程。

虽然自动内存管理机制有效地解决了内存泄漏问题，带来的代价是执行垃圾回收时会暂停进程，如果暂停的时间过长，程序看起来就像“卡死了”一样。

为什么在高并发下程序会卡死？

在理解了自动内存管理的基本原理后，我再带你分析一下，为什么在高并发场景下，这种自动内存管理的机制会更容易触发进程暂停。

一般来说，我们的微服务在收到一个请求后，执行一段业务逻辑，然后返回响应。这个过程中，会创建一些对象，比如说请求对象、响应对象和处理中间业务逻辑中需要使用的一些对象等等。随着这个请求响应的处理流程结束，我们创建的这些对象也就都没有用了，它们将会在下一次垃圾回收过程中被释放。

你需要注意的是，直到下一次垃圾回收之前，这些已经没有用的对象会一直占用内存。

那么，虚拟机是如何决定什么时候来执行垃圾回收呢？这里面的策略非常复杂，也有很多不同的实现，我们不展开来讲，但是无论是什么策略，如果内存不够用了，那肯定要执行一次垃圾回收的，否则程序就没法继续运行了。

在低并发情况下，单位时间内需要处理的请求不多，创建的对象数量不会很多，自动垃圾回收机制可以很好地发挥作用，它可以选择在系统不太忙的时候来执行垃圾回收，每次垃圾回收的对象数量也不多，相应的，程序暂停的时间非常短，短到我们都无法感知到这个暂停。这是一个良性的循环。

在高并发的情况下，一切都变得不一样了。

我们的程序会非常繁忙，短时间内就会创建大量的对象，这些对象将会迅速占满内存，这时候，由于没有内存可以使用了，垃圾回收被迫开始启动，并且，这次被迫执行的垃圾回收面临的是占满整个内存的海量对象，它执行的时间也会比较长，相应的，这个回收过程会导致进程长时间暂停。

进程长时间暂停，又会导致大量的请求积压等待处理，垃圾回收刚刚结束，更多的请求立刻涌进来，迅速占满内存，再次被迫执行垃圾回收，进入了一个恶性循环。如果垃圾回收的速度跟不上创建对象的速度，还可能会产生内存溢出的现象。

于是，就出现了我在这节课开始提到的那个情况：一到大促，大量请求过来，我们的服务就卡死了。

高并发下的内存管理技巧

对于开发者来说，垃圾回收是不可控的，而且是无法避免的。但是，我们还是可以通过一些方法来降低垃圾回收的频率，减少进程暂停的时长。

我们知道，只有使用过被丢弃的对象才是垃圾回收的目标，所以，我们需要想办法在处理大量请求的同时，尽量少的产生这种一次性对象。

最有效的方法就是，优化你的代码中处理请求的业务逻辑，尽量少的创建一次性对象，特别是占用内存较大的对象。比如说，我们可以把收到请求的 Request 对象在业务流程中一直传递下去，而不是每执行一个步骤，就创建一个内容和 Request 对象差不多的新对象。这里面没有多少通用的优化方法，你需要根据我告诉你的这个原则，针对你的业务逻辑来想办法进行优化。

对于需要频繁使用，占用内存较大的一次性对象，我们可以考虑自行回收并重用这些对象。实现的方法是这样的：我们可以为这些对象建立一个对象池。收到请求后，在对象池内申请一个对象，使用完后再放回到对象池中，这样就可以反复地重用这些对象，非常有效地避免频繁触发垃圾回收。

如果可能的话，使用更大内存的服务器，也可以非常有效地缓解这个问题。

以上这些方法，都可以在一定程度上缓解由于垃圾回收导致的进程暂停，如果你优化的好，是可以达到一个还不错的效果的。

当然，要从根本上来解决这个问题，办法只有一个，那就是绕开自动垃圾回收机制，自己来实现内存管理。但是，自行管理内存将会带来非常多的问题，比如说极大增加了程序的复杂度，可能会引起内存泄漏等等。

流计算平台 Flink，就是自行实现了一套内存管理机制，一定程度上缓解了处理大量数据时垃圾回收的问题，但是也带来了一些问题和 Bug，总体看来，效果并不是特别好。因此，一般情况下我并不推荐你这样做，具体还是要根据你的应用情况，综合权衡做出一个相对最优的选择。

小结

现代的编程语言，大多采用自动内存管理机制，虚拟机会不定期执行垃圾回收，自动释放我们不再使用的内存，但是执行垃圾回收的过程会导致进程暂停。

在高并发的场景下，会产生大量的待回收的对象，需要频繁地执行垃圾回收，导致程序长时间暂停，我们的程序看起来就像卡死了一样。为了缓解这个问题，我们需要尽量少地使用一次性对象，对于需要频繁使用，占用内存较大的一次性对象，我们可以考虑自行回收并重用这些对象，来减轻垃圾回收的压力。

思考题

如果我们的微服务的需求是处理大量的文本，比如说，每次请求会传入一个 10KB 左右的文本，在高并发的情况下，你会如何来优化这个程序，来尽量避免由于垃圾回收导致的进程卡死

问题？欢迎你在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (41)

linqw
2019-08-22

尝试回答下课后习题，老师有空帮忙看下哦
如果有一个微服务是处理大量的文本，感觉这种一般不会要求时延，大部分都会进行异步处理，更加注重服务的吞吐率，服务可以在更大的内存服务器进行部署，然后把新生代的eden设置的更大些，因为这些文本处理完不会再拿来复用，朝生夕灭，可以在新生代Minor GC，防止对象晋升到老年代，防止频繁的Major GC，如果晋升的对象过多大于老年代的连续内存空间也会有触发Full Gc，然后在这些处理文本的业务流程中，防止频繁的创建一次性的大对象，把文本对象做为业务流程直接传递下去，如果这些文本需要复用可以将他保存起来，防止频繁的创建。也为了保证服务的高可用，也需对服务做限流、负载、兜底的一些策略。

作者回复: 思路非常清晰，赞👍

110

leslie
2019-08-22

一路跟着老师学到现在我大致明白了老师想阐述什么或者说上次回答我的困惑时的答案了；其实老师是想传授：为何要用消息队列、如何使用、何种场景下使用其涉及什么知识我们应当如何把握它的使用。

老师上次的回答提到程序不用太深：不过其实程序、网络还有今天的课程提及的内存管理–其实是计算机组成原理的东西，如何合理的去结合这些知识才是消息队列把握好的关键；就像老师今天留的题目其实就是需要程序的垃圾回收机制的知识和组成原理的内存管理的知识结合才能给出相应的正确答案，不知道是否可以这样理解老师今天的题目？

同时在跟几位老师的课一起学习知识并梳理自己从业多年的知识体系：至少让我觉得之前对于课程的选择是正确的，至少从大的方面去理解了；老师其实是在授之与渔，而非简单的授之与鱼。

期待老师的下节课：希望老师解答一下我对于问题方向上的理解是否正确，谢谢。

作者回复: 我希望更给大家的，既能有鱼，先填饱肚子解决手上的问题，然后还能有渔，学到捕鱼的技能，受用终生。


36

Peter
2019-10-28

简单整理下jvm的一些概念，帮大家回忆回忆这些理论哈哈
垃圾回收算法：
标记清除：效率较低，会产生内存碎片
复制算法：将内存一分为二，通过不断将活着的对象移动到内存另一面，再清除这面，解决了效率低、内存碎片的问题，引来新的问题：内存一分为二代价太高
标记-整理算法：先标记（过程跟标记清除一样）再将存活对象都向一端移动，清理掉端边界以外的内存。适用于老年代
分代收集算法：将内存划分为几块，新生代采用复制算法，老年代采用标记-整理算法
垃圾收集器：
Serial收集器：新生代采用复制算法，会stop the world；老年代采用标记-整理算法，也会stop the world
ParNew收集器：Serial收集器的多线程版本，其他一模一样
Parallel Scavenge收集器：特点：可控制的吞吐量
CMS收集器：特点：重视服务响应速度，降低GC停顿时间
大致分为4个步骤
初始标记
并发标记
重新标记
并发清除
会在初始标记和重新标记这两步stop the world
G1收集器：特点：可预测的停顿，可以明确指定在一个长度为M毫秒的时间片段内，消耗在GC上的时间不得超过N毫秒
G1的运作大致分为以下几步：
初始标记
并发标记
最终标记
筛选回收
会在初始标记、最终标记、筛选回收时stop the world


36

钱

2019-08-23

课后思考及问题

1：这个算法有一个最大问题就是，在执行标记和清除过程中，必须把进程暂停，否则计算的结果就是不准确的。这也就是为什么发生垃圾回收的时候，我们的程序会卡死的原因。后续产生了许多变种的算法，这些算法更加复杂，可以减少一些进程暂停的时间，但都不能完全避免暂停进程。

对于这段有几个问题?

1-1：进程必须暂停，是在标记阶段还是在清除阶段？还是两者都会?

1-2：进程暂停这个实现过程是怎样的？暂停后需要再启动，这个又是一个怎样的过程?

1-3：后面解释进程必须暂停的原因是为了使计算结果更加准确，我觉得好比打扫卫生，我一个房间一个房间来，也不耽误其他房间的事，是不是暂停是不必须的，其实 young gc 几乎不停的在发生，只有发生full gc 的时候性能才会大大降低?

1-4：内存清除这个动作具体是怎么实现的？是电平复位？还是打上可以继续使用的标位？如果打标位这个该怎么打呢？一位一位的打？还是一个字节一个字节的打？更或者是一块一块的打?

作者回复: A1： 标记阶段需要暂停，清除阶段一般是不需要的。

A2：这个问题有点复杂，你可以参考一下：https://stackoverflow.com/questions/16558746/what-mechanism-jvm-use-to-block-threads-during-stop-the-world-pause

A3：对于GC来说只有一个房间，你是没有办法分成多个完全独立的小房间的。 像java中的young gc就是为了缓解这个问题，而产生的变种算法，它可以减少FullGC的次数，但没有办法完全避免FullGC。

A4：内存是按页为单位管理的，也就是一块一块的，对于JVM来说，它有一套复杂的数据结构来记录它管理的所有页面与对象引用之间的关系。所谓清除和移动对象，就是修改这个记录关系的数据结构。

共 3 条评论>

18

亚洲舞王.尼古拉斯赵...

2019-08-22

通过一个对象池，池子里的对象大小是10k，每次请求申请对象，结束请求归还对象。另外提个意见，老师能在每次新课的时候讲述一下上一课提的问题的答案吗?

共 5 条评论>

18

Jxin
2019-08-29

需求：
1.处理10kb的文本（文本存储和业务处理）
2.进程不卡死（高响应服务不中断）

分析：
1.请求不可丢失，响应要快，允许业务处理有一定滞后性。
2.发挥单机性能极限，但不越限导致卡顿或中断。

方案：
1.采用生产消费模式，接受到数据直接持久化，异步消费。为了实现数据积压，不丢失，可以走跨进程的实现，比如mq。
2.压测文本落盘和文本业务处理两个进程的负载能力，调整其接入层线程池线程数，找到最高单机并发上限，设置令牌桶做限流。通过水平扩提高总请求负载力。落盘和业务处理两块根据并发负载力调整相对集群的节点比例。

注：除非业务真有性能需求，不然千万别一个对象传到底。架设防腐层，业务解耦，对系统的扩展力很是重要。对于高速发展的项目（变化大且快），其价值远大于这么点性能提升。（先往易于演变的架构走，前期堆机器。真到业务量足够庞大，需要调优时再调优）

共 1 条评论 >　　18

lupguo
2020-05-09

高并发和避免gc，尽可能少的系统调用次数，让用户态的应用程序可以快速接受tcp传过来的数据（增大接收套接字的buffer缓冲区大小，可以降低用户态和内核态的内存拷贝频次，降低上下文切换开销）。
全业务处理流程考虑用指针传递，避免内存拷贝或者堆上内存开销（栈上开销os自行回收），降低被gc可回收的变量基数。
考虑业务处理线程或协程去复用一些申请的内存区域，比如go中的buffer pool，以及通过buffer reset在处理完业务时候自动释放，可以复用申请的内存区域。
通过pprof，runtime去监控和观察内存和、gc的实际情况做对照，了解应用程序实际内存的使用情况。
暂时想到这么多。

作者回复: 👍👍👍

13

a、
2019-08-22

通过jstat 观察gc情况和分析gc日志，来合理分配堆内存，年轻代，年老代大小，尽量让对象在minor gc就能被回收，而不需要执行full gc。因为full gc执行速度慢，程序暂停时间就长

9

努力努力再努力
2019-11-21

老师，假如我有一个对象，这个对象作为接口的入参，但是前端在传值的时候，只传了部分字段，那么在申请内存空间的时候，这个对象是只申请传值了的这些对象所以占用的空间，还是所有属性占用的空间

作者回复: 这个问题比较复杂，没有统一的答案，在不同的编程语言中处理都不太一样。

一般面向对象的语言，比如Java这种，对象的属性如果是基本类型，它的内存空间会随着对象的创建就占用上了。如果属性也是一个对象，什么时候真正去申请内存，取决于你的代码中是在什么时候new出这个属性对象的。

4

渔村蓝
2019-08-22

拿到文本，异步写入硬盘，给队列一个路径，另一个线程监控队列，一个个路径拿出来加载到内存一个个处理。

3

vi
2019-08-22

1. 使用对象池，重复使用对象，每次处理文件中的行数据，更新到利用的对象中
2. 把对象外迁，使用中间件缓存对象，不被GC扫描到

共 1 条评论>

3

冰激凌的眼泪
2019-08-22

占用内存差不多的，是不是比较适合池化？

作者回复: 是这样的。


3

业余草
2019-08-22

除了方法论，还想要一个结合方法论的demo实现！

共 1 条评论>


3

笑傲流云
2019-08-23

老师，说下我的思路：1，jvm对字符串有优化，字符串是不可变对象，通过字符串常量池，可以复用一些字符串；2，文本10kb过大，是否可以拆分？建议分割文本，形成小对象直接在年轻代被垃圾回收，避免大对象直接进入老年代，引发频繁的full gc；3，Kafka底层存储机制大量使用了page cache，把子节码缓存在磁盘，避免大量对象引发gc问题。


1

许童童
2019-08-22

我会考虑使用享元模式，预先分配10KB 左右的对象池，当请求进来时，从对象池中拿一个来使用，用完后，自己释放，以此来自己回收，复用这些对象，减少对象的创建，从而减少垃圾回收。


2

徐李
2022-06-21

关于问题：
1.增大硬件
2.当请求达到一定数目时，直接返回请求超时，就是不到后台，在网关处就返回


1

石佩
2020-03-27

目前这些知识点对于消息队列来说都是必须的，每一点甚至最后在消息队列中都有单独的模块进行实现，文末留的问题就是消息队列实际面对的问题，Broker会接收各种消息进行文本处理 然后存储

1

九黎曦

2020-03-26

内存管理这一块比较烦的是，对象移动问题，因为对象需要移动，所以对象所有的引用都要修改，而这个过程中需要暂停，而且有些对象不支持移动，例如系统的对象等，所以内存管理的最佳策略是分析服务的对象周期制定内存区分配方案，如果参差不齐可以考虑拆分，这也是拆分微服务的策略之一

共 2 条评论>

1

成都小郭

2019-09-19

disruptor,一个高性能队列,也有这样对对象的优化处理..初始化队列长度之后,就会创建那么多个对象,新消息进来就给对象赋值,减少了很多对象创建和销毁的时间

1

木小柒

2019-08-22

接上一条，点击空白留言出去了，这个功能好尴尬。可以先看机器能给到的内存量和cpu消耗，看大约一秒钟可以处理多少文件。然后限流，可以把文件存本地，也可以存消息队列中，看资源来定。控制文件数量，虽然处理排队慢了，但不至于挂掉。

作者回复: 那能否用本节课中学到的一些内存管理的方法来解决呢?

1