xingdong_2017

java进程和普通进程在内存管理上区别于联系

Linux与JVM的内存关系分析

在这篇文章中：

引言

在一些物理内存为8g的服务器上，主要运行一个Java服务，系统内存分配如下：Java服务的JVM堆大小设置为6g，一个监控进程占用大约600m，Linux自身使用大约800m。从表面上，物理内存应该是足够使用的；但实际运行的情况是，会发生大量使用SWAP(说明物理内存不够使用了)，如下图所示。同时，由于SWAP和GC同时发生会致使JVM严重卡顿，所以我们要追问：内存究竟去哪儿了？

要分析这个问题，理解JVM和操作系统之间的内存关系非常重要。接下来主要就Linux与JVM之间的内存关系进行一些分析。

一、Linux与进程内存模型

JVM以一个进程（Process）的身份运行在Linux系统上，了解Linux与进程的内存关系，是理解JVM与Linux内存的关系的基础。 下图给出了硬件、系统、进程三个层面的内存之间的概要关系。

昵称：xingdong_2017
园龄：6年9个月
粉丝：0
关注：1
+加关注

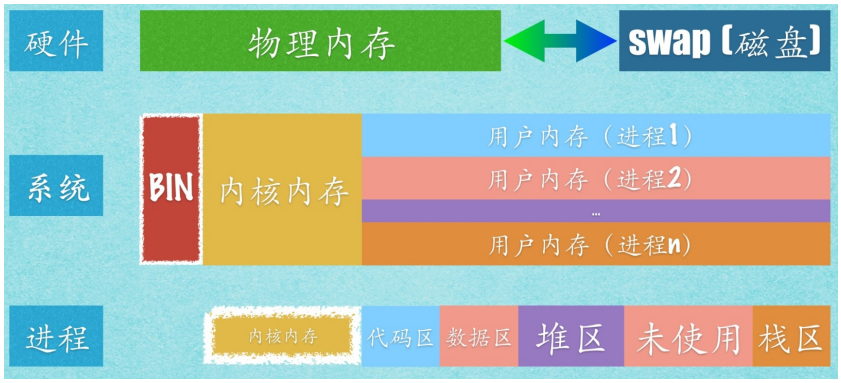
< 2023年10月 >

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

搜索
找找看

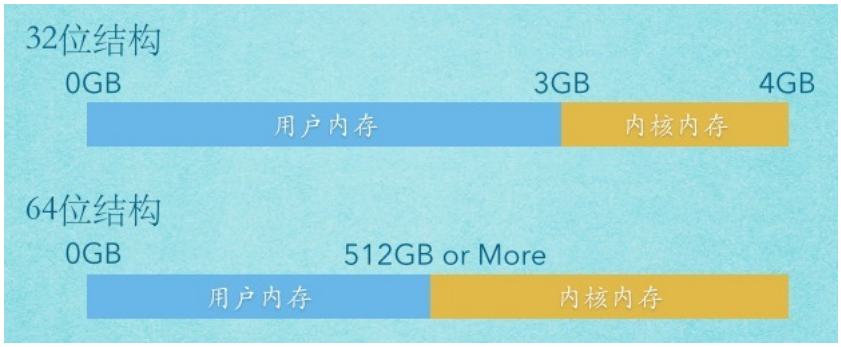
常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

文章分类
netty源码(2)



从硬件上看，Linux系统的内存空间由两个部分构成：物理内存和SWAP（位于磁盘）。物理内存是Linux活动时使用的主要内存区域；当物理内存不够使用时，Linux会把一部分暂时不用的内存数据放到磁盘上的SWAP中去，以便腾出更多的可用内存空间；而当需要使用位于SWAP的数据时，必须先将其换回到内存中。

从Linux系统上看，除了引导系统的BIN区，整个内存空间主要被分成两个部分：内核内存（Kernel space）、用户内存（User space）。 内核内存是Linux自身使用的内存空间，主要提供给程序调度、内存分配、连接硬件资源等程序逻辑使用。用户内存是提供给各个进程主要空间，Linux给各个进程提供相同的虚拟内存空间；这使得进程之间相互独立，互不干扰。实现的方法是采用虚拟内存技术：**给每一个进程一定虚拟内存空间，而只有当虚拟内存实际被使用时，才分配物理内存。**如下图所示，对于32的Linux系统来说，一般将0～3G的虚拟内存空间分配做为用户空间，将3～4G的虚拟内存空间分配为内核空间【每一个进程都冗余3G-4G的虚拟内核空间】；64位系统的划分情况是类似的。



从进程的角度来看，进程能直接访问的用户内存（**虚拟内存空间**）被划分为5个部分：代码区、数据区、堆区、栈区、未使用区。代码区中存放应用程序的机器代码，运行过程中代码不能被修改，具有只读和固定大小的特点。数据区中存放了应用程序中的全局数据，静态数据和一些常量字符串等，其大小也是固定的。堆是运行时程序动态申请的空间，属于程序运行时直接申请、释放的内存资源。栈区用来存放函数的传入参数、临时变量，以及返回地址等数据。未使用区是分配新内存空间的预备区域。

二、进程与JVM内存模型

JVM本质就是一个进程，因此其内存模型也有进程的一般特点。但是，JVM又不是一个普通的进程，其在内存模型上有许多崭新的特点，

主要原因有两个：

1.JVM将许多本来属于操作系统管理范畴的东西，移植到了JVM内部，目的在于减少系统调用的次数；**【这也是netty中分配heap内存比分配direct内存快的原因、不需要系统调用】**

2. Java NIO，目的在于减少用于读写IO的系统调用的开销。 JVM进程与普通进程内存模型比较如下图:

需要说明的是，这个模型的并不是JVM内存使用的精确模型，更侧重于从操作系统的角度而省略了一些JVM的内部细节（尽管也很重要）。下面从用户内存和内核内存两个方面讲解JVM进程的内存特点。

1.用户内存

上图特别强调了JVM进程模型的代码区和数据区指的是JVM自身的，而非Java程序的。普通进程栈区，在JVM一般仅仅用做线程栈。JVM的堆区和普通进程的差别是最大的，下面具体详细说明：

首先是永久代。**永久代本质上是Java程序的代码区和数据区。Java程序中类（class），会被加载到整个区域的不同数据结构中去，包括常量池、域、方法数据、方法体、构造函数、以及类中的专用方法、实例初始化、接口初始化等。这个区域对于操作系统来说，是堆的一个部分；而对于Java程序来说，这是容纳程序本身及静态资源的空间，使得JVM能够解释执行Java程序。**

其次是新生代和老年代。新生代和老年代才是Java程序真正使用的堆空间，主要用于内存对象的存储；但是其管理方式和普通进程有本质的区别。 普通进程在运行时给内存对象分配空间时，比如C++执行new操作时，会触发一次分配内存空间的系统调用，由操作系统的线程根据对象的大小分配好空间后返回；同时，程序释放对象时，比如C++执行delete操作时，也会触发一次系统调用，通知操作系统对象所占用的空间已经可以回收。JVM对内存的使用和一般进程不同。JVM向操作系统申请一整段内存区域（具体大小可以在JVM参数调节）作为Java程序的堆（分为新生代和老年代）；当Java程序申请内存空间，比如执行new操作，JVM将在这段空间中按所需大小分配给Java程序，并且Java程序不负责通知JVM何时可以释放这个对象的空间，垃圾对象内存空间的回收由JVM进行。【也就说说jvm的堆是jvm进程启动的时候提前在虚拟内存中划分好的一块内容】

JVM的内存管理方式的优点是显而易见的，包括：

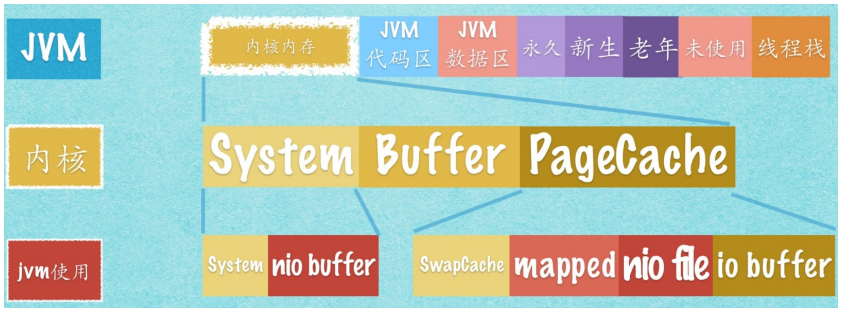
第一，减少系统调用的次数，JVM在给Java程序分配内存空间时不需要操作系统干预，仅仅在Java堆大小变化时需要向操作系统申请内存或通知回收，而普通程序每次内存空间的分配回收都需要系统调用参与；

第二，减少内存泄漏，普通程序没有（或者没有及时）通知操作系统内存空间的释放是内存泄漏的重要原因之一，而由JVM统一管理，可以避免程序员带来的内存泄漏问题。

最后是未使用区，未使用区是分配新内存空间的预备区域。对于普通进程来说，这个区域被可用于堆和栈空间的申请及释放，每次堆内存分配都会使用这个区域，因此大小变动频繁；对于JVM进程来说，调整堆大小及线程栈时会使用该区域，而堆大小一般较少调整，因此大小相对稳定。操作系统会动态调整这个区域的大小，并且这个区域通常并没有被分配实际的物理内存，只是允许进程在这个区域申请堆或栈空间。

2.内核内存

应用程序通常不直接和内核内存打交道，内核内存由操作系统进行管理和使用；不过随着Linux对性能的关注及改进，一些新的特性使得应用程序可以使用内核内存，或者是映射到内核空间。Java NIO正是在这种背景下诞生的，其充分利用了Linux系统的新特性，提升了Java程序的IO性能。

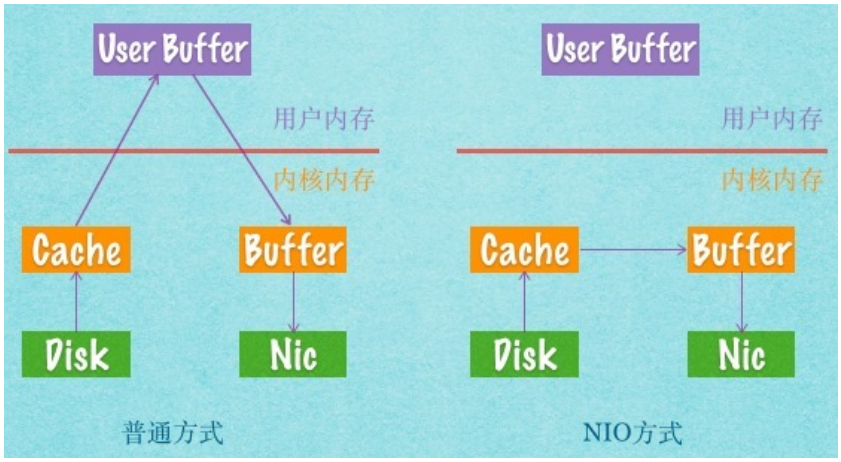


上图给出了Java NIO使用的内核内存在linux系统中的分布情况。nio buffer主要包括：nio使用各种channel时所使用的ByteBuffer、Java程序主动使用ByteBuffer.allocateDirector申请分配的Buffer。而在PageCache里面，nio使用的内存主要包括：FileChannel.map方式打开文件占用mapped、FileChannel.transferTo和FileChannel.transferFrom所需要的Cache（图中标示 nio file）。

通过JMX可以监控到NIO Buffer和 mapped 的使用情况，如下图所示。不过，FileChannel的实现是通过系统调用使用原生的PageCache，过程对于Java是透明的，无

法监控到这部分内存的使用大小。

Linux和Java NIO在内核内存上开辟空间给程序使用，主要是减少不要的复制，以减少IO操作系统调用的开销。例如，将磁盘文件的数据发送网卡，使用普通方法和NIO时，数据流动比较下图所示：



将数据在内核内存和用户内存之间拷贝是比较消耗资源和时间的事情，而从上图我们可以看到，【通过直接 在内核虚拟内存部分开辟空间、这样就不需要内核和用户空间的复制、减少了上下文切换的开销】通过NIO的方式减少了2次内核内存和用户内存之间的数据拷贝。这是Java NIO高性能的重要机制之一（另一个是异步非阻塞）。

从上面可以看出，内核内存对于Java程序性能也非常重要，因此，在划分系统内存使用时候，一定要给内核留出一定可用空间。

三、案例分析

1.内存分配问题

通过上面的分析，省略比较小的区域，可以总结JVM占用的内存： JVM内存 ≈ Java永久代 + Java堆(新生代和老年代) + 线程栈 + Java NIO

回到文章开头提出的问题，原来的内存分配是：6g(java堆) + 600m(监控) + 800m(系统)，剩余大约600m内存未分配。

现在分析这600m内存的分配情况：

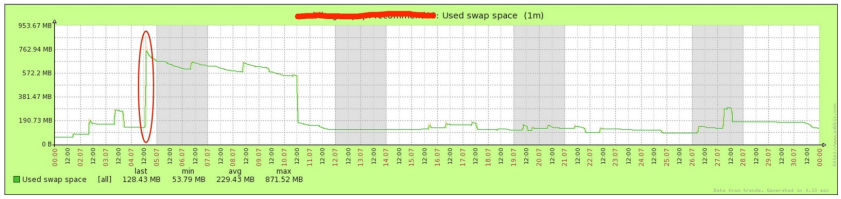
(1)Linux保留大约200m，这部分是Linux正常运行的需要，

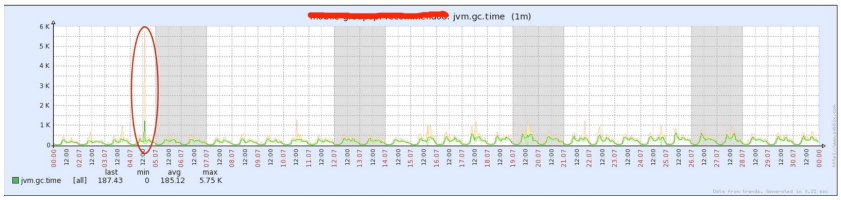
(2)Java服务的线程数量是160个，JVM默认的线程栈大小是1m，因此使用160m内存，

(3)Java NIO buffer，通过JMX查到最多占用了200m，

(4)Java服务使用NIO大量读写文件，需要使用PageCache【将磁盘文件缓存在物理内存中，java的NIO读取文件是利用directBuffer在进程的虚拟地址空间分配内存来装载磁盘文件】，正如前面分析，这个暂时不好定量估算大小。 前三项加起来已经560m，因此可以断定Linux物理内存不够使用。

细心的人会发现，引言中给出两个服务器，一个SWAP最多占用了2.16g，另外一个SWAP最多占用了871m；但是，似乎我们的内存缺口没有那么大。事实上，这是由于SWAP和GC同时进行造成的，从下图可以看到，SWAP的使用和长时间的GC在同一时刻发生。





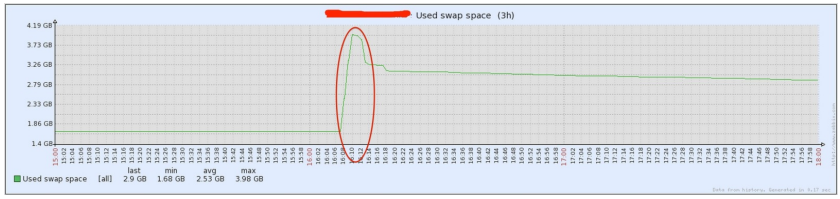
SWAP和GC同时发生会导致GC时间很长，JVM严重卡顿，极端的情况下会导致服务崩溃。原因如下：JVM进行GC时，时需要对相应堆分区的已用内存进行遍历；假如GC的时候，有堆的一部分内容被交换到SWAP中，遍历到这部分的时候就需要将其交换回内存，同时由于内存空间不足，就需要把内存中堆的另外一部分换到SWAP中去；于是在遍历堆分区的过程中，(极端情况下)会把整个堆分区轮流往SWAP写一遍。Linux对SWAP的回收是滞后的，我们就会看到大量SWAP占用。

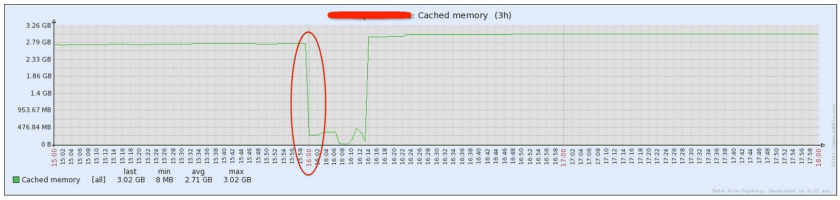
上述问题，可以通过减少堆大小，或者增加物理内存解决。

因此，我们得出一个结论：部署Java服务的Linux系统，在内存分配上，需要避免SWAP的使用；具体如何分配需要综合考虑不同场景下JVM对Java永久代 、Java堆(新生代和老年代)、线程栈、Java NIO所使用内存的需求。

2.内存泄漏问题

另一个案例是，8g内存的服务器，Linux使用800m，监控进程使用600m，堆大小设置4g；系统可用内存有2.5g左右，但是也发生了大量的SWAP占用。 分析这个问题如下：(1)在这个场景中， Java永久代 、Java堆(新生代和老年代)、线程栈所用内存基本是固定的，因此，占用内存过多的原因就定位在Java NIO上。 (2)根据前面的模型，Java NIO使用的内存主要分布在Linux内核内存的System区和PageCache区。查看监控的记录，如下图，我们可以看到发生SWAP之前，也就是物理内存不够使用的时候，PageCache【PageCache说明】急剧缩小。因此，可以定位在System区的Java NIO Buffer发生内存泄漏。





(3)由于NIO的DirectByteBuffer需要在GC的后期被回收，因此连续申请DirectByteBuffer的程序，通常需要调用System.gc()，避免长时间不发生FullGC导致引用在old区的DirectByteBuffer内存泄漏。分析到此，可以推断有两种可能的原因：第一，Java程序没有在必要的时候调用System.gc()；第二，System.gc()被禁用。 (4)最后是要排查JVM启动参数和Java程序的DirectByteBuffer使用情况。在本例中，查看JVM启动参数，发现启用了-XX:+DisableExplicitGC导致System.gc()被禁用。

四、总结

本文详细分析了Linux与JVM的内存关系，比较了一般进程与JVM进程使用内存的异同点，理解这些特性将对Linux系统内存分配、JVM调优、Java程序优化有帮助。限于篇幅关系仅仅列举两个案例，希望起到抛砖引玉的作用。

参考

1. http://www.ibm.com/developerworks/library/j-nativememory-linux/
2. http://www.ibm.com/developerworks/library/l-kernel-memory-access/
3. http://www.ibm.com/developerworks/library/j-zerocopy/
4. 《深入分析Java Web技术内幕》

分类: netty源码

posted on 2018-10-19 11:15 xingdong_2017 阅读(253) 评论(0) 编辑 收藏 举报