java直接内存与堆内存

NIO的Buffer提供了一个可以不经过JVM内存直接访问系统物理内存的类——DirectBuffer。 DirectBuffer类继承自ByteBuffer，但和普通的ByteBuffer不同，普通的ByteBuffer仍在JVM堆上分配内存，其最大内存受到最大堆内存的限制；而DirectBuffer直接分配在物理内存中，并不占用堆空间，其可申请的最大内存受操作系统限制。

直接内存的读写操作比普通Buffer快，但它的创建、销毁比普通Buffer慢。

因此直接内存使用于需要大内存空间且频繁访问的场合，不适用于频繁申请释放内存的场合。

（Note：DirectBuffer并没有真正向OS申请分配内存，其最终还是通过调用Unsafe的allocateMemory()来进行内存分配。不过JVM对Direct Memory可申请的大小也有限制，可用-XX:MaxDirectMemorySize=1M设置，这部分内存不受JVM垃圾回收管理。）

以下是一些测试：

代码：

复制代码

1 class DirectMemory {

2

3 // 分配堆内存

4 public static void bufferAccess() {

5 long startTime = System.currentTimeMillis();

6 ByteBuffer b = ByteBuffer.allocate(500);

7 for (int i = 0; i < 1000000; i++) {

8 for (int j = 0; j < 99; j++)

9 b.putInt(j);

10 b.flip();

11 for (int j = 0; j < 99; j++)

12 b.getInt();

13 b.clear();

14 }

15 long endTime = System.currentTimeMillis();

16 System.out.println("access\_nondirect:" + (endTime - startTime));

17 }

18

19 // 直接分配内存

20 public static void directAccess() {

21 long startTime = System.currentTimeMillis();

22 ByteBuffer b = ByteBuffer.allocateDirect(500);

23 for (int i = 0; i < 1000000; i++) {

24 for (int j = 0; j < 99; j++)

25 b.putInt(j);

26 b.flip();

27 for (int j = 0; j < 99; j++)

28 b.getInt();

29 b.clear();

30 }

31 long endTime = System.currentTimeMillis();

32 System.out.println("access\_direct:" + (endTime - startTime));

33 }

34

35 public static void bufferAllocate() {

36 long startTime = System.currentTimeMillis();

37 for (int i = 0; i < 1000000; i++) {

38 ByteBuffer.allocate(1000);

39 }

40 long endTime = System.currentTimeMillis();

41 System.out.println("allocate\_nondirect:" + (endTime - startTime));

42 }

43

44 public static void directAllocate() {

45 long startTime = System.currentTimeMillis();

46 for (int i = 0; i < 1000000; i++) {

47 ByteBuffer.allocateDirect(1000);

48 }

49 long endTime = System.currentTimeMillis();

50 System.out.println("allocate\_direct:" + (endTime - startTime));

51 }

52

53 public static void main(String args[]) {

54 System.out.println("访问性能测试：");

55 bufferAccess();

56 directAccess();

57

58 System.out.println();

59

60 System.out.println("分配性能测试：");

61 bufferAllocate();

62 directAllocate();

63 }

64 }

复制代码

结果：

复制代码

访问性能测试：

access\_nondirect:157

access\_direct:134

分配性能测试：

allocate\_nondirect:231

allocate\_direct:613

复制代码

可见与在JVM堆分配内存(allocate)相比，直接内存分配（allocateDirect）的访问性能更好，但分配较慢。（一般如此，当然数据量小的话差别不是那么明显）